Janne Johansson

New User Interface Architecture for NetWiser Product with AngularJS

Helsinki Metropolia University of Applied Sciences

Master's Degree

Information Technology

Master's Thesis

30 April 2015

Helsinki Metropolia University of Applied Sciences

Author	Janne Johansson
Title	New User Interface Architecture for NetWiser Product with AngularJS
Number of Pages	59 pages + 1 appendix
Date	30 April 2015
Degree	Master of Engineering
Degree Programme	Information Technology
Instructor	Ville Jääskeläinen, Principal Lecturer

The topic of the thesis was to develop a new user interface for a product called NetWiser. The need for a new user interface originated from the problems experienced with the technology of the old user interface. A decision was made to replace the old Java applets based user interface with a new JavaScript based user interface. In practice, the technology switch meant a complete redevelopment of the user interface application component. A complete redesign of the user interface layout was excluded from the scope of the thesis. The thesis was also limited to cover only one of the many NetWiser user interfaces.

The research consisted of two parts. In the first part a JavaScript framework was selected among 48 JavaScript frameworks and libraries as the technology to develop the new NetWiser user interface. The research compared JavaScript frameworks based on basic and technological project requirements. At the end, five most suitable JavaScript frameworks were studied by comparing an example application source code developed with each of the frameworks. Finally, AngularJS was selected as the most suitable framework.

The second part of the research studied the best practices related to AngularJS application development. The findings of the research were utilized in the development of the new NetWiser user interface. The development consisted of setting up a development environment with proper tools, replacing the used technology and improving some of the user interface components.

The selection of AngularJS as the development framework was a success. AngularJS provides a comprehensive set of features for web application development. The community behind AngularJS is big enough to support professional web application development. The new NetWiser user interface was developed using AngularJS. The application functionality was successfully verified using automated and manual tests. Feedback from the organization stakeholders and a customer was positive.

Keywords	AngularJS, JavaScript, User interface, Single page application

Tekijä Otsikko Sivumäärä Aika	Janne Johansson New User Interface Architecture for NetWiser Product with AngularJS 59 sivua + 1 liite 30.4.2015
Tutkinto	Master of Engineering
Koulutusohjelma	Information Technology
Ohjaaja	Ville Jääskeläinen, Principal Lecturer

Opinnäytetyön aiheena on uuden käyttöliittymän kehittäminen NetWiser tuotteelle. Käyttöliittymän kehittämisen tarve sai alkunsa havaituista ongelmista tuotteen vanhan käyttöliittymän teknologian kanssa. Vanha Java applet -pohjainen käyttöliittymä päätettiin korvata uudella JavaScript-pohjaisella käyttöliittymällä. Teknologianvaihto tarkoitti koko käyttöliittymäsovelluksen uudelleenkehitystä. Käyttöliittymän ulkoasun rakenteen uudelleensuunnittelu päätettiin jättää pois opinnäytetyöstä. Opinnäytetyötä rajattiin sisällyttämällä vain yhden NetWiser käyttöliittymän uudelleenkehitys.

Opinnäytetyön tutkimus koostuu kahdesta osasta. Ensimmäisessä osassa valitaan JavaScript-sovelluskehys 48 JavaScript-sovelluskehyksen ja -kirjaston joukosta uudeksi NetWiser tuotteen käyttöliittymä teknologiaksi. Tutkimus vertailee JavaScript-sovelluskehyksiä projektivaatimusten näkökulmasta. Lisäksi viiden sopivimman JavaScript-sovelluskehyksen avulla kehitetyn esimerkkisovelluksen lähdekoodeja vertaillaan toisiinsa. Lopuksi AngularJS valittiin sopivimmaksi sovelluskehykseksi NetWiser käyttöliittymäprojektiin.

Tutkimuksen toisessa osassa selvitettiin AngularJS-sovelluskehyksen parhaita käytäntöjä sovelluskehityksessä. Tutkimuksen tuloksia käytettiin uuden NetWiser käyttöliittymän kehitysprojektissa. Kehitysprojekti koostuu projektiympäristön asentamisesta kunnollisten työkalujen avulla, teknologianvaihdosta sekä muutamien käyttöliittymäkomponenttien vaihdosta nykyaikaisiin versioihin.

AngularJS-sovelluskehyksen valinta käytettäväksi teknologiaksi osoittautui onnistuneeksi. AngularJS tarjoaa laajan valikoiman ominaisuuksia Internet-sovellusten kehittämiseen. AngularJS-sovelluskehyksen parissa toimiva yhteisö on riittävän laaja tukeakseen ammattimaista sovelluskehitystä. Uusi NetWiser käyttöliittymä kehitettiin käyttäen AngularJS-sovelluskehystä. Sovelluksen toimivuus varmistettiin onnistuneesti automaattisten sekä manuaalisten testien avulla. Palaute asiakkaalta ja yrityksen sidosryhmältä oli positiivista.

Asiasanat	AngularJS, JavaScript, Käyttöliittymä, Single page application

Contents

List of Figures

1 Introduction

A vast number if IT (information technology) systems have been developed over the past decades. Many of the older IT systems become obsolete or get replaced by other IT systems. IT systems that are used for decades must be improved to answer to the evolving user experience expectations and technological challenges.

A common problem in the industry is an aging IT system providing a service crucial to its users. The system may be running in an environment that is quickly becoming obsolete or the user experience provided by the system developed decades earlier is no longer accepted by today's users. Especially the user experience expectations have evolved a lot over time. Some old IT systems are so difficult to use that to perform simple operations, reading a user's manual is required. The need to create responsive user interfaces has increased greatly. Long waiting periods between operations is no longer accepted. This thesis covers a development of a new user interface to an existing IT system resolving some of the problems related to old IT systems.

1.1 Case Company and Technology Challenge for the Study

The case company of this study is CGI (Consultants to Government and Industry) Suomi Oy. CGI Suomi Oy is part of the global CGI Group Inc. CGI provides IT and business process services with the experience of 68,000 professionals in 40 countries.

The case company has an IT software product called NetWiser. NetWiser is used to collect and analyse telecommunications network data. The purpose of the Master's Thesis was to develop a new web user interface to the NetWiser product using a suitable JavaScript framework. The development project is part of a bigger undertaking to further develop various parts of the product.

Two central problems have been identified in the current NetWiser user interface. The first problem is the technology used for developing it. Part of the user interface has been developed using Java applet technology. Java applets have had a large number of security vulnerabilities in the past years causing many web browser to repel the technology by default. Oracle has provided several fixes for the security issues in Java

technology over time resulting in multiple software update cycles in customer organizations. To replace Java applet technology, some parts of the application must be redeveloped completely.

The second problem is the usability and overall look and feel of the user interface. A number of user interface usability issues have been identified by the users. The user interface was developed several years ago and as a result the look and feel does not meet the current standards.

1.2 Study Objective

The purpose of the Master's Thesis was to develop a new web user interface to the NetWiser product using a suitable JavaScript framework. The development project is part of a bigger project to further develop various parts of the NetWiser product.

The outcome of the project is a new web user interface application that replaces the old one in the commercial NetWiser product. It should be noted that the study concentrates more on the structure of the application rather than on how a specific user interface component is created. The study will also provides the guidelines for the practical development work.

The Master's thesis project concentrates on the technology switch and on some enhancements to the user interface. The basis of the new interface relies heavily on the current user interface. The final version of the new user interface including company branding is excluded from the scope of the thesis. The information provided by this study will be used in the development of the new NetWiser user interface. JavaScript as a language is known to be prone for creating applications with a messy code. A well-defined structure of the application is important for the future development work.

1.3 Methodology

To reach the objective, the thesis first explores viable JavaScript frameworks used for the development of a new NetWiser user interface. The second part of the research investigates the best development practices of the chosen technology conforming to the NetWiser user interface requirements.

The JavaScript framework study was conducted in multiple phases using a process of elimination, following a funnel logic. In the first phase, the frameworks were compared against the basic requirements of the NetWiser user interface. The requirements include commonly used corporation browser Internet Explorer 8 support, activeness of the development of the chosen framework and developer availability in Finland. A search on a few dozen JavaScript libraries and frameworks was conducted to see whether the requirements are met. The data of the details on each framework was collected from the Internet.

In the second phase, the selected frameworks were checked for technological requirements. Technological requirements include the architecture of the application structure provided by the framework and JavaScript language as the development language. In the final phase, the details of each remaining framework were evaluated in order to select the framework used in the new NetWiser user interface development. Finally, the best practices of the selected framework were researched to create a well-structured application with the framework.

The web user interface was developed as a SPA (single page application) to follow the current standards of responsive web applications. The backend server communication is limited to data transfer only. Reduced number of server calls allows the application to operate smoothly without constant waiting periods.

The framework testability features for unit testing and end-to-end testing were also researched. Testing of the application was conducted using automated unit tests and automated end-to-end tests. In addition, manual user interface tests were conducted to ensure quality. A continuous integration system was used to run the automated tests after every committed change to the versioning system.

1.4 Scope and Structure

The thesis covers a full software development project including a background technology study, selection of the technology, best practices research, setting up the development environment, development work and testing. Some of the concepts included in the thesis are relevant in many software development projects and some are specific to JavaScript web application development using the selected framework.

Section 2 describes the current NetWiser product architecture and user interface design. Section 3 describes the Internet technologies background explaining existing technologies and concentrating on JavaScript in more detail. Section 4 covers the research part of the thesis JavaScript framework selection. Section 5 describes the development project of the NetWiser user interface using best practices and section 6 includes the summary and conclusions of the thesis.

2 Current Solution

NetWiser is a platform for collecting and analysing data from CSP's (Communication Service Provider) networks. NetWiser analytics provides a view to many areas of the CSP's networks. The analytics data can be used in revenue assurance, fraud management, network planning, service assurance and regulatory compliancy. Some typical use cases include detecting and preventing abuse, investigating a problem in end customer's service and providing data retention required by the law. [18]

2.1 User Interface Architecture

The current NetWiser user interface is built with a combination of Java applet, HTML and JavaScript. Figure 1 illustrates the NetWiser architecture in a broad component level. The user interface presented in the figure is one of the many user interfaces built using the same architecture. The difference between various user interfaces is the data that is represented. For example one user interface presents data on the amount of traffic in the network and another user interface is used to view individual data records of the collected network events. The data backend in the figure is described in a very general level not including any specific information on how interfaces are built or what kind of components are included.

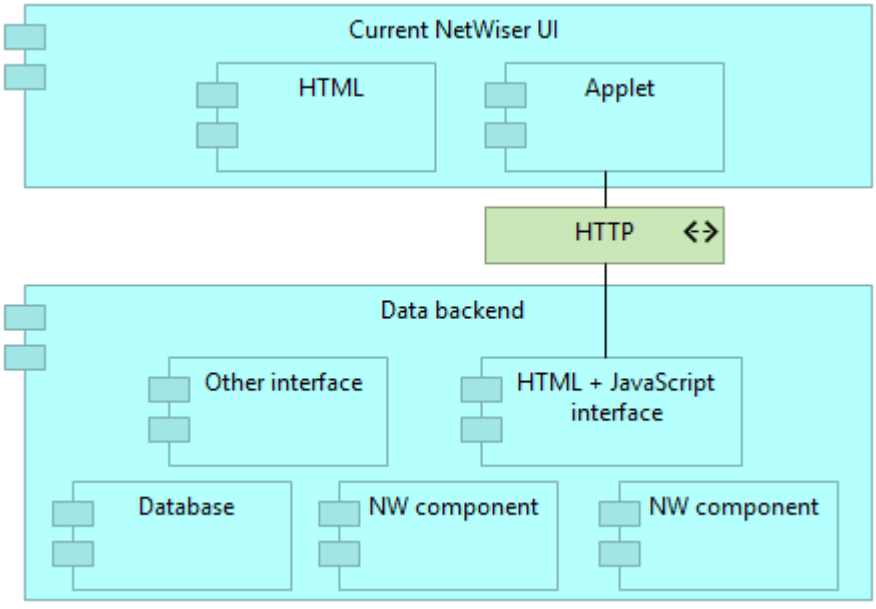

Figure 1. Current NetWiser component architecture

The component developed with Java applet technology provides communication with the data backend and the application business logic. HTML is used the view the data query results from the backend. The data backend provides the data using HTTP protocol in HTML, JavaScript, images and raw textual data formats. HTML and images are partly generated dynamically on the server side. HTML is mostly used to present the data in a table format and images present the data over time as bar or line diagrams.

The flow on the user interface on a typical use case consists of the following steps: After the user has logged in to the system, the selected user interface is loaded including static HTML and a Java applet. Java applet loads the content of pre-defined queries and the content of selection menus from the data backend using a raw textual interface. Next, the user can select a pre-defined query or construct a new query using various selections provided by the Java applet part of the user interface. Once the user triggers the search, the applet queries the data backend according to the user selections. The query results are presented on the data view part of the user interface as HTML and images retrieved from the backend. JavaScript is used to allow the user to view different aspects of the query result.

2.2 User Interface Design

NetWiser user interface has been designed to provide useful data in a short amount of time. The ability to view the desired data has surpassed the user experience aspects in importance. The user interface box model layout is illustrated in Figure 2. The upper part “Search parameters” is the area that displays currently selected search parameters. The area is rendered by the applet. The middle part of the figure illustrates the data view frames. The number and layout of data view frames varies between different user interfaces. Data view frame content consist of dynamic HTML and images. The navigation bar is at the bottom of the layout. The navigation bar content is dynamic HTML. The query parameters are given in a separate window provided by the applet.

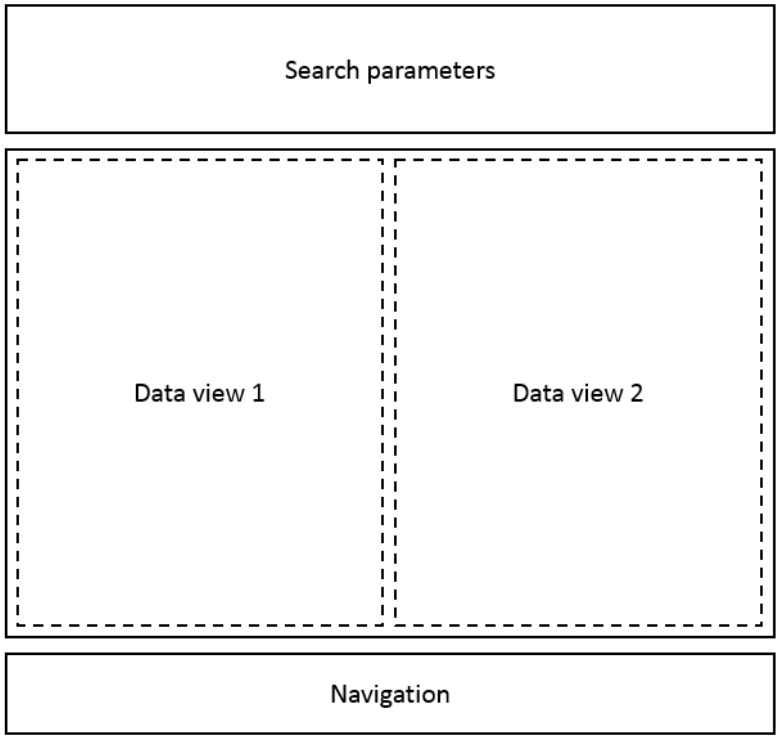

Figure 2. User interface box model layout

Figure 3 illustrates an example screenshot of one of the user interface views called One Glance Desktop or OGD. OGD displays physical or virtual network element measurements over a selected time period.

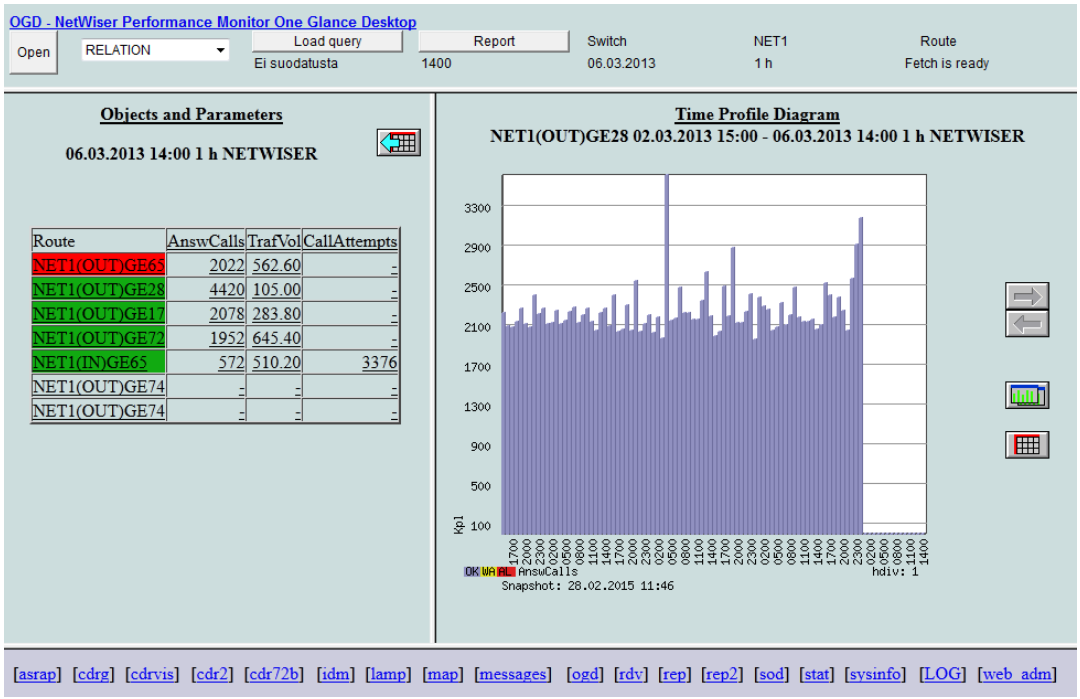

Figure 3. NetWiser One Glance Desktop screenshot

On the left in the data view frame the measurements are presented over a one hour time period in a table format. On the right the number of answered calls in the first network element are represented as a bar diagram over a four day time period. Each bar represents the number of answered calls in one hour. On the top part of the figure some of the query parameters are displayed. The buttons on the top are used to manage the query creation and execution.

2.3 Problems with Current Architecture and Design

One of the biggest architectural problems in the NetWiser user interface is the Java applet technology used. The user is required to constantly update workstation Java version due to the many updates fixing various security vulnerabilities. Browsers detect old Java versions the on user's workstation and refuse to display any Java applet content until the Java version is updated. In practice this means that users are at times unable to use the service as if the service itself was unavailable. Replacing the Java applet technology was selected as the main focus on the project covered in this thesis.

One design problem is the multi-window user interface. Some users like to have multiple user interfaces open at the same time and can get confused when all user interfaces open more than one window. The user interface is designed for expert users. Many values displayed on the screen have no explanation and their meaning may not be obvious to first time users or users that use the interface only once in a while. These design issues are also addressed in the thesis project.

3 Internet Technologies Background

The Internet technologies background section describes various Internet technologies used in client side web application development. The latter part of the section describes JavaScript technology and especially JavaScript frameworks in more detail. JavaScript is the main technology used in the new NetWiser user interface development.

3.1 Internet Browser Application Technologies

Internet web pages are displayed using web browsers and are developed using a standardized language HTML (HyperText Markup Language). Web pages can be static or dynamic. Static web pages always display the same content whereas dynamic pages contain changing content. Web applications are applications that employ the web browser as a client, and as such, require dynamic web pages to present the changing application state. The HTML language itself has a limited support for dynamic content changes requiring additional browser technologies for application development.

Different browser application technologies can be divided into two categories based on the running conditions. The first category includes technologies that are natively supported by most or all Internet browsers. The second category includes technologies that require additional runtime environment and a browser plug-in to run. Most of the second category technologies are available on PC and MAC computers but have limited or no support on smart phones and tablet computers.

JavaScript

JavaScript is a web application technology supported natively by all modern browsers. JavaScript is advertised as “the programming language of the web” [1]. JavaScript browser applications are completely run on the client side whereas the server side is primarily used for data access. In some web application scenarios the server is also used to generate HTML pages dynamically based on the application state. Server generated HTML pages can be combined with the capabilities of JavaScript. JavaScript is supported on PC and MAC computers as well as smart phones and tablet computers.

Java applets

Java applets is a web browser technology by Oracle. Java applets allow similar user experience as Java desktop applications with restrictions to the access of the running system resources. To run Java applets, Java Runtime Environment and a browser Java plug-in are required on the client system.

Over the last few years there has been several security problems with Java applets. The security issues have caused browsers to reject Java by default and to display security warnings. Figure 4 represents a security warning on Oracle website even with the latest Java Runtime Environment installed on the user workstation.

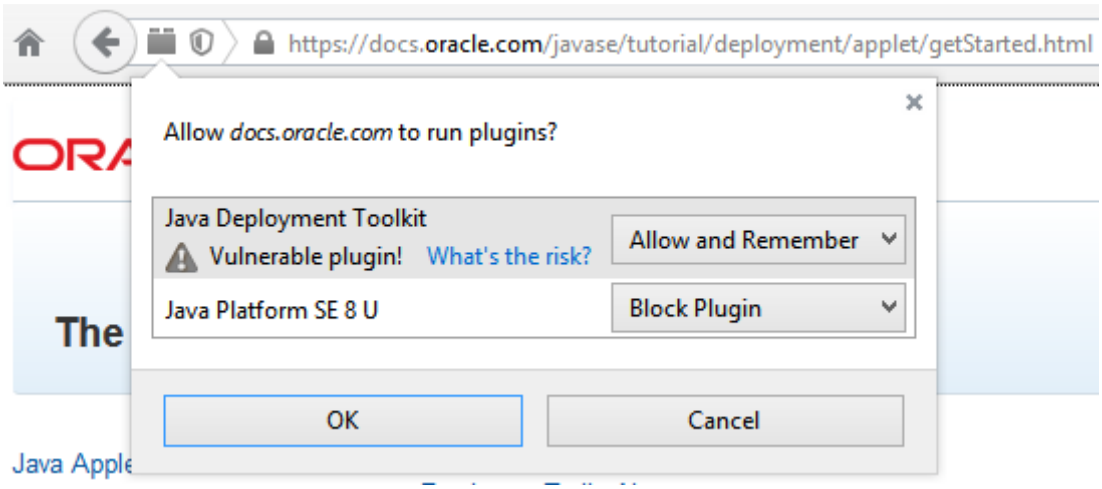

Figure 4. Java security warning in Firefox web browser with latest JRE installed

Hypothetically there could be incidents preventing the usage of business-critical web applications in corporations. Browser refuses to start a Java applet as a result of an expired Java version as the IT department has not yet updated Java Runtime to the latest version on all workstations.

Adobe Flash

Adobe Flash is a technology commonly used to view video and animation content on the web. Many browser based games and video players are developed with Flash. After the HTML5 native support for video and animation the demand for technologies such as Flash has decreased. Flash requires Flash Player and Flash browser plug-in installed on the client system to run Flash applications.

Microsoft Silverlight

Microsoft Silverlight has been originally developed as a competing technology for Adobe Flash. Browser and operating system support has been more limited on Silverlight compared to Adobe Flash. Silverlight requires Silverlight runtime and browser plug-in to run. Figure 5 illustrates Firefox browser on Netflix website using Silverlight plug-in.

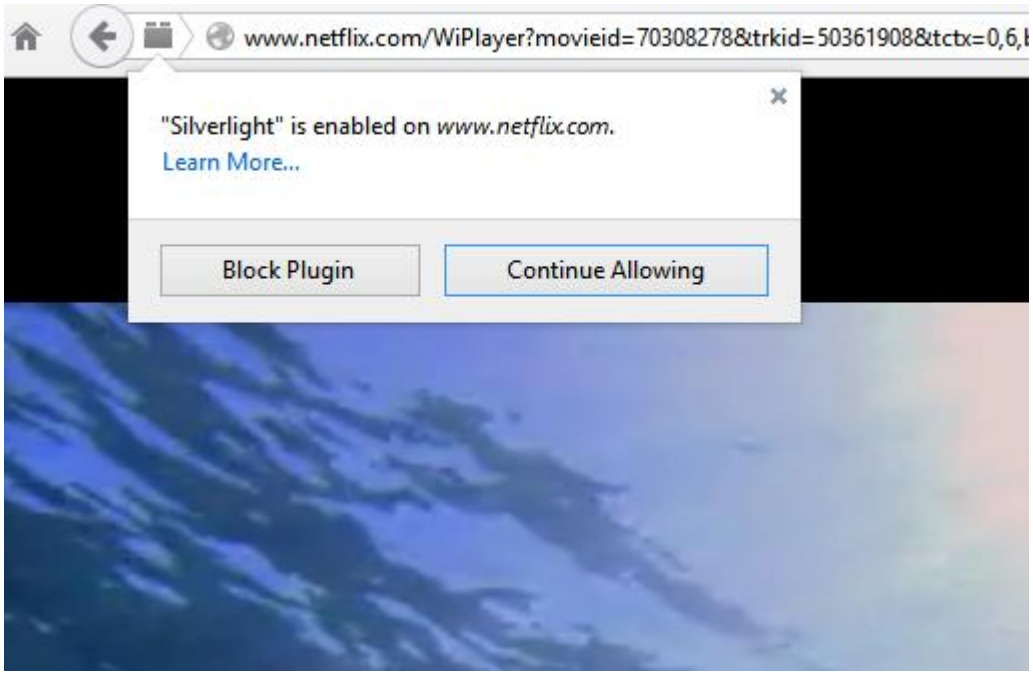

Figure 5. Netflix using Silverlight plugin on Firefox browser

The end of life to the latest version 5 of Silverlight has been set to October 10th 2021 [2]. No plans for further versions have been announced.

3.2 JavaScript in Internet Browsers

JavaScript was announced December 4th 1995 by Netscape Communication Corporation and Sun Microsystems [3]. The original press release is available in the Internet Archive. According to the press release, JavaScript was designed for creating web applications that utilise server and client resources providing multimedia richness. The technology was hyped as open, cross-platform scripting language. The plan to standardize the language was already mentioned in the press release. JavaScript support was released as part of Netscape Navigator 2.0 beta at the same time.

Microsoft released JavaScript as part of Internet Explorer 3.0 beta May 29th 1996 [4]. Microsoft's implementation of JavaScript is called JScript. JScript supports integration with Microsoft's ActiveX browser technology which is not part of JavaScript or the ECMAScript standard. There are also some other differences in the implementation of JavaScript and JScript.

The JavaScript standard is called ECMAScript. JavaScript and JScript are implementations of ECMAScript. Instead of ECMAScript, JavaScript is the commonly known name of the language and also the name referenced in the literature. The first version of ECMAScript was standardized by Ecma International in June 1997 [5]. The latest 5.1 edition of the standard is from June 2011 [6].

Today JavaScript is supported in all modern web browsers including Microsoft's Internet Explorer, Apple's Safari, Mozilla's Firefox, Google's Chrome and Opera browser. JavaScript implementations on browsers on smart phones and tablet devices also exist. To overcome differences in various implementations and browser versions a number of JavaScript libraries and frameworks have been developed by other parties. Using a JavaScript library or framework simplifies the development of web applications by a great degree. In theory a developer does not need to be familiar with all the intricacies of different JavaScript and browser versions. A function call on a JavaScript library produces a similar effect on each of the modern day browsers.

3.3 JavaScript Frameworks

JavaScript framework is an application framework for web application development. There are two interesting comparisons related to JavaScript frameworks. First is the comparison to JavaScript libraries and the second is the comparison to server side web applications frameworks.

A dictionary definition for application frameworks helps to understand the difference between JavaScript libraries and JavaScript frameworks:

> "*An application framework is a software library that provides a fundamental structure to support the development of applications for a specific environment. An application framework acts as the skeletal support to build an application.*" [9]

The main difference between a software library and a software framework is the lack of provided application structure in the former. A JavaScript framework is a JavaScript library that provides a structure and common tools for creating web applications. The provided application structure in many modern day JavaScript frameworks is MVC (Model-View-Controller) or MVC-like architectural pattern. The common tools include data binding, page routing, templating and custom HTML elements. [8: p.13]

A web application developed with a server side web application framework differs from a web application developed with a JavaScript framework. The main difference is the location and runtime environment of the application logic. On a server side application, the application logic resides and is run in the server whereas a JavaScript application is downloaded and run on the user's web browser. A JavaScript application requires less server and network capacity. As most of the application work is done on the web browser and there is no need for constant server requests, a JavaScript application often appears to be faster. Figure 6 illustrates the difference between a server side application and a JavaScript application page transition.

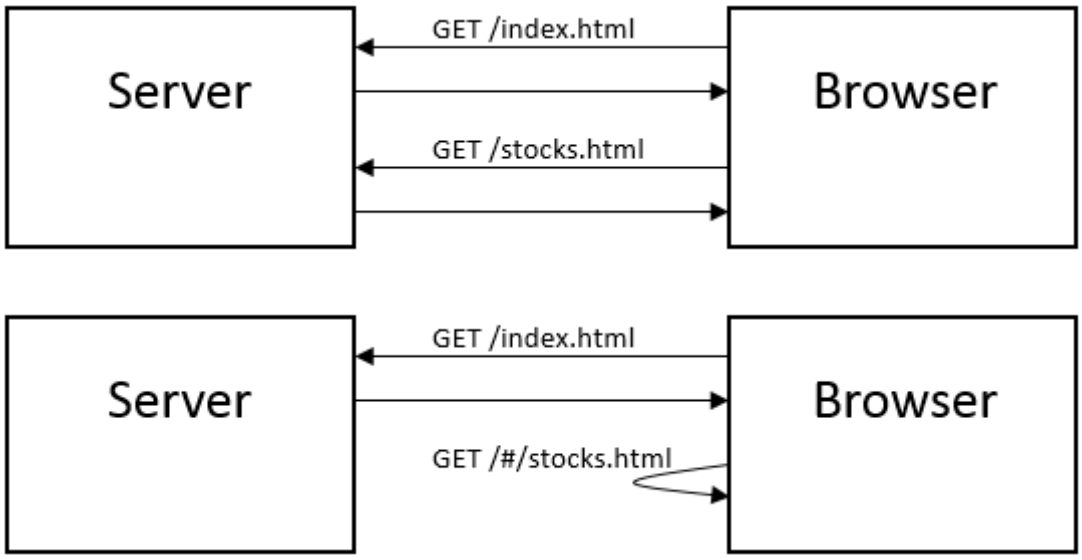

Figure 6. Server side application and JavaScript application page transition

In the upper part of the figure the server side application calls the server every time when the user navigates between pages. The entire page is loaded on the web browser. In the lower part of the figure a JavaScript application calls the server once in the beginning and loads the entire application to the user's web browser. After the initial

load, the page transition is handled entirely on the web browser. Ideally the JavaScript loads only the changed parts of the page on the browser.

3.3.1 MVC and MVVM Architectural Patterns in JavaScript Frameworks

JavaScript frameworks often provide application structure by implementing MVC (Model-View-Controller) or MVVM (Model-View-ViewModel) architectural pattern. MVC is an old pattern from earlier decades and MVVM is a later derivation of MVC. The idea behind MVC is the separation of different components used in application development to individual modules. The modules can be developed separately as the interaction between modules is limited. It should be noted that the implementation of both of these patterns vary in different technologies.

The Model in MVC represents the data model of an application. The Model is completely unaware of the user interface. In many MVC implementations the Model objects are very simple data encapsulation objects.

The View is the user interface seen by the user. All user interactions are triggered on the View level. The View presents the current state of the Model objects. As the Model changes, the View receives a change event in order to update the View.

The Controller contains the application's business logic and overall decision making relating to what is shown on the screen. User interactions from the View are forwarded to the Controller. The Controller also acts as the link to the underlying data storage system. A Controller and a View pair is often used to create a reusable user interface component. [15, 7: p.2-3]

The ViewModel in MVVM pattern replaces the Controller of MVC. The ViewModel binds the Model and the View enabling two-way data binding between the modules. When data changes in one of the modules the other is automatically updated with the new value. Similarly to the Controller, the ViewModel is used to receive user interactions from the view and handle the data transactions. [16: p.4-5]

3.3.2 JavaScript Framework Features

JavaScript frameworks provide a set of features where some of the features are same or similar between different frameworks. This section covers the key features provided by many JavaScript frameworks.

DOM Tree Manipulation

DOM (Document Object Model) tree is the object structure that HTML or other markup languages are used to describe. The root element is `<html>` that contains `<head>`, `<body>` and other elements and their attributes in a tree like structure. JavaScript frameworks provide functions for finding elements in the DOM tree and manipulating their content. DOM tree manipulation is a very handy way to change what is visible to the user. Many of the latest JavaScript frameworks manipulate DOM tree automatically based on the changes in the data model. Automatic DOM tree manipulation avoids the direct DOM tree manipulation by the developers.

Two-Way Data Binding

Two-way data binding is a feature that greatly reduces the amount of JavaScript code required for copying object values from one place to another. A common application scenario is to have a model object in the background JavaScript code and a representation of the model object in HTML user interface. Technically the background object and the user interface object are two different entities. Two-way data binding binds the background object to the user interface object. Any change to the background object data is automatically copied to the user interface object and vice-versa. Two-way data binding is an important feature to avoid direct DOM tree manipulation. [7: p.6]

URL Routing

URL routing is an extension to server side URL routing where page `http://www.example.com/index.html` provides different content than page `http://www.example/page.html`. Client side URL routes are called hashbang URLs. Calling a hashbang URL does not make a server call when called after the first time. For example a first call to `http://www.example.com/index.html#page1` generates a

server call but transitioning to page `http://www.example.com/index.html#page2` is handled only by the client side JavaScript framework. The JavaScript framework URL routing can be configured to provide different page for `#page1` and `#page2` calls. [7: p.139-142]

HTML Templating

HTML templating is a feature that provides a way to create repeating dynamic sections to HTML. For example a list of users creates the same layout structure for every user with different data content. The mechanic to provide HTML templating varies between frameworks. Some frameworks provide custom HTML elements and element attributes for improved templating.

Dependency Injection

Dependency injection provides a way to declare dependencies between application modules. This allows independent module development and a clear way to separate responsibilities between modules. For example a data module could be dependent on data parser module. The dependency is declared in the data module. The dependency injection ensures that the dependent data parser module is available and is provided to the data module. [7: p.73-74]

4 Selecting JavaScript Framework

The first part of the thesis research was about selecting a JavaScript framework for the implementation of the new NetWiser web user interface. Selection of any framework is a trade-off. The purpose of this section is to help understand what benefits or drawbacks different frameworks have. The goal was not to select the best JavaScript framework in the world but to select the most suitable framework in the given context. A JavaScript framework that addresses web application development as a whole, is preferable to a JavaScript library that addresses individual problems in web application development. A framework offers a structure and modularity to the application whereas using a library would require creating an application structure as part of the project.

4.1 Selection Requirements

Three basic requirements were selected that the framework must fulfil. The requirements are based on the surrounding conditions rather than any functional specifications. In addition to the basic requirements, few technical requirements were selected. The technical requirements originate from the overall application architecture and justifiable preferences.

The NetWiser product has been developed over the past twenty years. As the product has a long lifecycle, also the selected framework must be supported long into the future. The future is unknown, but it is possible to set a measureable requirement to assess development longevity by looking into the past. Based on the lifecycle need, framework development activity was selected as the first basic requirement. The activity was measured by having at least three years of continuous active development. The requirement does not give a full picture of a specific framework's activity, but it helps in elimination of frameworks that are too recent or inactive.

A web user interface is used using a web browser. Active frameworks quickly adopt changes introduced by new web browsers, but the older web browsers may have limited support. Existing customers' oldest default web browser was Internet Explorer 8, hence the native support for Internet Explorer 8 was selected as the second basic re-

quirement. Naturally, the application was tested with other commonly used web browsers.

The availability of developers for the framework in Finland was selected as the third basic requirement. For products with long lifecycles it is important to select a framework that is commonly known in order to be able to find developers in the future. The developer availability – or current demand – was checked using job market web sites for open jobs.

An architectural application structure provided by the framework was selected as the first technical requirement is. Having a ready application architectural structure means that the developers' work can be mostly concentrated in creating the application business logic rather than thinking about how the application core is developed. JavaScript as the framework programming language was selected as the second technical requirement. Some frameworks use one language for application development and compile the result into JavaScript. HTML as the application layout provider was selected as the third technical requirement. Some frameworks are developed entirely with JavaScript and create the view seen by users dynamically. The use of HTML is preferable to allow developers to comprehend the layout structure quickly.

4.2 Selection Method

The selection of the JavaScript framework was carried out by a method of elimination from a large group of JavaScript frameworks and libraries. As there were dozens of frameworks included in the selection process, it would have been very time consuming to closely examine each framework in detail. To save time, the selection process was divided into three phases. In each phase the frameworks were examined using different criteria. After each phase a number of frameworks were eliminated and only the remaining frameworks were examined in the following phase. After the final phase, one framework was selected to be used in the implementation of the new NetWiser web user interface. Each phase is described in detail in the following chapters.

The initial set of JavaScript frameworks and libraries were selected based on Google searches, common knowledge and various lists found in the Internet. This arbitrary method of gathering data may have resulted in some frameworks to be missed from the list. However, all key frameworks and libraries are well known and as one of the

criteria was to use well-established framework, the risk of missing an important framework was considered to be low.

During the development phase some unforeseen need or problem could have arisen that was not solvable by the selected JavaScript framework. For this reason, using other JavaScript libraries in the development was not ruled out by the selection process.

4.3 Basic Requirements

A total of 48 JavaScript frameworks and libraries were selected for the first comparison phase. JavaScript libraries were also included because it may not be apparent which implementation offers an application framework and which is considered to be a library. The basic requirements were project development activity, support for Internet Explorer 8 and availability of developers in Finland. The reasoning for these requirements is described in Chapter 4.1. The table with the full comparison results is included in Appendix 1. The data was collected at November 17th 2014.

The activity of each framework and library project was checked from the related website. Many of the projects are open source and are hosted in GitHub (https://github.com/) code hosting service. The link to the project's GitHub repository can usually be found in the project website. On the GitHub repository front page there is a link "Releases" listing all project releases with release dates. For projects not hosted in GitHub the project activity was checked from the project version history in the project website. From the 48 projects only PhoneJS's project activity status was unclear. 23 projects were found out to be too recent or inactive.

The support for IE8 (Internet Explorer 8) was checked from the project documentation for all projects. Two projects AngularJS and jQuery have dropped IE8 support in the latest version. However, both projects pass the IE8 support requirement because bug fix releases are continued for the latest version that supports IE8. Seven projects' support for IE8 cannot be determined from the project documentation or other available sources. None of the seven projects passed the activity requirement. Six projects have no support for IE8.

The third requirement was the availability of developers in Finland. The developer availability was checked using job market websites monster.fi, oikotie.fi, duunitori.fi, te-

palvelut.fi and linkedin.com using the framework name as the search criteria and Finland as area. When the same job was found in multiple websites, it counts as one. The job status is checked for 22 projects that passed both the activity and IE8 support requirements. At least one job advertisement was required to pass the open jobs requirement. Two projects with most open jobs were AngularJS and jQuery with more than twenty available jobs.

A total of eight frameworks and libraries passed all three basic requirements: AngularJS, Backbone.js, Dojo toolkit, Ember.js, Enyo, Google Web Toolkit, jQuery and Knockout. After the first selection phase, most of the frameworks and libraries were eliminated.

4.4 Technical Requirements

The second selection phase was to exclude technologies that do not provide application structure architecture or use another programming language for application development. The reasons for eliminating some of the libraries are discussed in the following paragraphs.

Enyo is a JavaScript framework advertised as a framework for creating web applications for multiple platforms. In addition to desktop browsers Enyo, supports mobile browsers and packaged apps in phones, tablets and smart TV's. Enyo is developed entirely with JavaScript. In HTML there is only one placeholder element to which Enyo builds the entire application. Enyo does not conform to the requirement to use HTML for layout creation.

GWT (Google Web Toolkit) is a web application development toolkit. GWT based applications are developed using Java programming language. GWT compiler compiles Java source files to JavaScript. The produced application is run on the user web browser as a JavaScript application. GWT does not conform to the JavaScript language requirement, thus it is not suitable for the NetWiser user interface development project.

The most popular JavaScript library in the world is jQuery [10]. jQuery provides plenty of individual tools for web application development such as DOM tree manipulation and Asynchronous JavaScript and XML (AJAX) server communication. Although jQuery

provides such a rich set of web application development tools, it does not provide any kind of application structure architecture. For the lack of provided application structure architecture, jQuery was eliminated as a suitable solution for the NetWiser user interface development.

After the second selection phase, five frameworks AngularJS, Backbone.js, Dojo toolkit, Ember.js and Knockout conformed to all basic and technical requirements. The remaining frameworks provide application structure by employing MVC or MVVM (Model-View-ViewModel) architectural pattern. All of the remaining frameworks have been developed using JavaScript programming language.

4.5 Deeper Analysis of the Most Suitable JavaScript Frameworks

After all but five JavaScript frameworks were eliminated, the remaining frameworks deserve a more comprehensive analysis on the way they are used in web application development. The idea of the analysis is to provide some sort of understanding on how applications are developed with the frameworks. Only the main framework features are covered in this thesis. In the following chapters any notable pros and cons of the frameworks are listed to better understand what kind of benefits or issues each framework has. A simple web application example is examined that has been developed with each of the frameworks. The purpose of the examination is to see how clear and concise the developed code and related HTML content is. Based on the previous phases it was clear that any of remaining frameworks can be used in the NetWiser user interface development project.

To avoid the actual application development with multiple frameworks, a JavaScript framework comparison web site TodoMVC (http://todomvc.com/) was used for the application examples. TodoMVC provides a simple to-do list application developed using many popular JavaScript frameworks. All of the selected frameworks and some additional ones are included in the website. The source code for the application examples is available at GitHub. Figure 7 illustrates a screen capture of the to-do list example application.

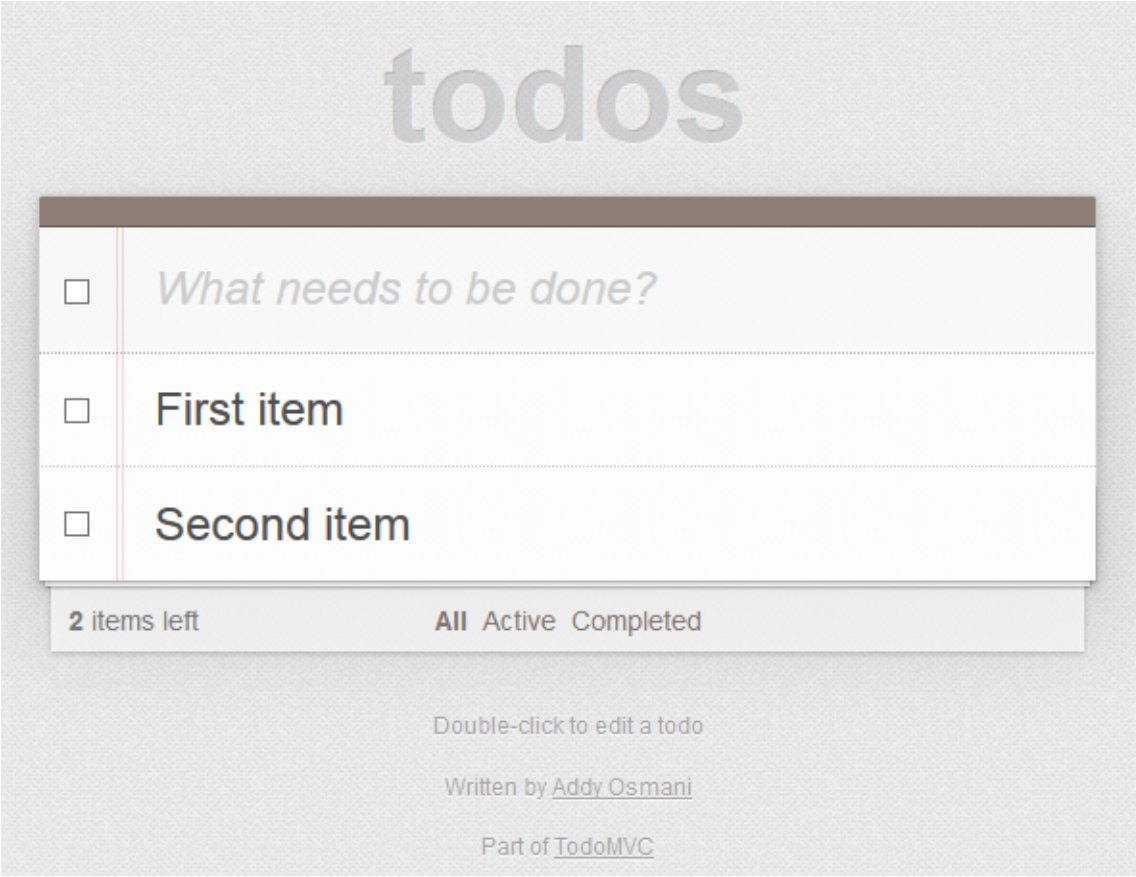

Figure 7. TodoMVC example application

In the example application user can add, remove and edit items in the to-do list. User can also mark items as completed. The first row is used to add new items. The rest of the rows illustrate the items already on the list. Filtering options to allow user to display only some of the items on the list are at the bottom of the figure. The version of the to-do list applications used in the comparison is 1.3 released in September 2014.

4.5.1 Backbone.js

Backcone.js was designed to provide structure for web applications. The main features include core components of the MVC pattern and declarative event system for views. Backbone.js is dependent on Underscore.js, another JavaScript library that provides useful JavaScript helper functions. Another dependency is a subset of jQuery that provides persistence using REST and browser history support. Other JavaScript libraries such as Lo-Dash and Zepto can be used instead of Underscore.js and jQuery [8: p.15-17, 12].

There is some controversy whether Backbone.js provides enough building blocks for web application development. Backbone.js does not provide two way-data binding and is lacking in memory management leaving the memory clean-up for the developer. Backbone.js view components manipulate the DOM tree directly. As a result unit testing view components becomes difficult [11]. As Backbone.js provides only few webapplication development features, the development team needs to choose other libraries to support the missing features. Additional work related to choosing component libraries is undesirable. In other type of project setting the freedom to choose component libraries could be seen as an advantage.

Examination of the example to-do list application is concentrated on the presentation of the first line “What needs to be done?” shown in Figure 7, the presentation of the existing lines and the action to add a new item to the list. The first line consists of a checkbox, a text input field and a title label. After user has inputted some text, the action to add a new item can be triggered by pressing the Enter key. The checkbox is used to toggle the selection of all of the items in the list.

Figure 8 illustrates a section of the *index.html* file of Backbone.js implementation of the to-do list application. Lines 9 to 25 under the `<section>` and `<footer>` elements is the layout skeleton for the entire application. Layout component details and functionality originate elsewhere. Looking on lines 9 to 20 which contains the main section of the application layout does not reveal much of the application details.

```
9   <section id="todoapp">
10      <header id="header">
11          <h1>todos</h1>
12          <input id="new-todo" placeholder="What needs to be done?" autofocus>
13      </header>
14      <section id="main">
15          <input id="toggle-all" type="checkbox">
16          <label for="toggle-all">Mark all as complete</label>
17          <ul id="todo-list"></ul>
18      </section>
19      <footer id="footer"></footer>
20  </section>
21  <footer id="info">
22      <p>Double-click to edit a todo</p>
23      <p>Written by <a href="https://github.com/addyosmani">Addy Osmani</a></p>
24      <p>Part of <a href="http://todomvc.com">TodoMVC</a></p>
25  </footer>
26  <script type="text/template" id="item-template">
27      <div class="view">
28          <input class="toggle" type="checkbox" <%= completed ? 'checked' : '' %>>
29          <label><%- title %></label>
30          <button class="destroy"></button>
31      </div>
32      <input class="edit" value="<%- title %>">
33  </script>
```

Figure 8. Backbone.js HTML snippet for to-do list application [21]

Line 12 in Figure 8 contains the `<input>` element for new to-do list item. The `<input>` checkbox to toggle selection of all items in the list is on line 15. The list element `<ul>` for existing to-do list items is on line 17. In lines 26 to 33 there is a `<script>` element template for items in the to-do list. Noticeable here is the separation of the list item template from the application section. The reader is required to look at two places in order to understand how the layout is structured and which HTML elements are used. In small application that is not a problem, but in a larger application the scattering of the layout structure becomes an issue.

A Backbone.js view is a central building block of the application that contains data model, event handling and DOM tree manipulation. Figure 9 illustrates the initialization of the application view events and components in `app-view.js` file. On line 32 the new to-do item `<input>` element with identifier `new-todo` is set to `$input` variable. In a similar way the `<ul>` element of to-do list items is assigned to the `$list` variable. On line 22 the key press event on the new to-do item `<input>` element is bound to a function called `createOnEnter`. The variable called `app` is the global root object for the Backbone.js application. The list of to-do list items is stored in the `app.todos` variable initialised in `todos.js` file. On line 37 a listener is added to the to-do list add event on function `addOne`.

```
20                  // Delegated events for creating new items, and clearing completed ones.
21                  events: {
22                          'keypress #new-todo': 'createOnEnter',
23                          'click #clear-completed': 'clearCompleted',
24                          'click #toggle-all': 'toggleAllComplete'
25                  },
26
27                  // At initialization we bind to the relevant events on the `Todos`
28                  // collection, when items are added or changed. Kick things off by
29                  // loading any preexisting todos that might be saved in *localStorage*.
30                  initialize: function () {
31                          this.allCheckbox = this.$('#toggle-all')[0];
32                          this.$input = this.$('#new-todo');
33                          this.$footer = this.$('#footer');
34                          this.$main = this.$('#main');
35                          this.$list = $('#todo-list');
36
37                          this.listenTo(app.todos, 'add', this.addOne);
38                          this.listenTo(app.todos, 'reset', this.addAll);
39                          this.listenTo(app.todos, 'change:completed', this.filterOne);
40                          this.listenTo(app.todos, 'filter', this.filterAll);
41                          this.listenTo(app.todos, 'all', this.render);
42
43                          // Suppresses 'add' events with {reset: true} and prevents the app view
44                          // from being re-rendered for every model. Only renders when the 'reset'
45                          // event is triggered at the end of the fetch.
46                          app.todos.fetch({reset: true});
47                  },
```

Figure 9. Backbone.js event and component initialization for to-do list [21]

Notable on Figure 9 is the way the event triggers and variables are tightly coupled to the HTML elements using identifiers. Tightly coupling is very common in JavaScript and HTML development. Tight coupling results from the way JavaScript is used to directly manipulate the DOM tree and locate HTML elements using identifiers. As a result of tight coupling, the code seen on Figure 9 is not reusable as a component on some other application. Also changes on HTML require changes to the JavaScript in many cases. Using loose coupling would require another technique or an additional layer of abstraction that is not provided by Backbone.js. Loose coupling is preferred in software development because changes in one component do not require changes in the loosely coupled component [13: p.40].

Figure 10 illustrates the event functions related to adding a new to-do item to the list. After user has entered some content to the new to-do list item input field and pressed Enter key, the function `createOnEnter` on line 108 is triggered. The event is triggered on every key press but the if-statement on line 109 checks that the provided key is Enter and some content has been inputted. The function creates a new to-do item, places the item in the storage variable and clears the user entered content. After the new to-do item has been added, the function `addOne` on line 78 is triggered. The function creates a

new `TodoView` view based on the Figure 8 line 26 to 33 HTML template. The `TodoView` template and other features are defined in *todo-view.js* file. The view's HTML representation is added under the `<ul>` list element using view's `render` function.

```
76                  // Add a single todo item to the list by creating a view for it, and
77                  // appending its element to the `<ul>`.
78                  addOne: function (todo) {
79                          var view = new app.TodoView({ model: todo });
80                          this.$list.append(view.render().el);
81                  },
82

105
106                 // If you hit return in the main input field, create new **Todo** model,
107                 // persisting it to *localStorage*.
108                 createOnEnter: function (e) {
109                         if (e.which === ENTER_KEY && this.$input.val().trim()) {
110                                 app.todos.create(this.newAttributes());
111                                 this.$input.val('');
112                         }
113                 },
```

Figure 10. Backbone.js add to-do item and key press event function [21]

Notable on Figures 9 and 10 is the linear execution of events. The code is easy to understand and easy to follow. The Backbone.js API seems logical enough to understand just by looking at someone else's code without prior experience with the API. Once a model element is created from the values retrieved from HTML – here a to-do list item – the developer is required to add item to the storage and create the HTML element with content from the template.

Backbone.js contains many usable features in web application development. From the first glimpse the learning curve seems low on understanding the different features of the Backbone.js API. However, the lack of two-way data binding and use of DOM tree direct manipulation result in the framework appear less appealing.

4.5.2 Dojo Toolkit

Dojo toolkit is one of the oldest JavaScript frameworks getting its first release in 2005. Dojo started more as a library providing useful tools in web application development. The framework-like features were added later in version 1.7 released in 2011. The Dojo toolkit structure is modular allowing a creation of custom builds that include only the required parts of the toolkit. [14]

Dojo toolkit example to-do list application in version 1.3 is several years old. For this reason the example to-do list application is taken from the master branch of the TodoMVC application set January 29th 2015.

Figure 11 illustrates the header part of to-do list application of creating new to-do list items and the section that lists the to-do list items. The creation of new to-do list is implemented on lines 34 to 36 using HTML form. The creation event is triggered using attribute `data-dojo-attach-event` definition by setting `addTodo` function call to the form's standard `submit` event.

The `<section>` element starting from line 38 is hidden when there are no items in the to-do list. The attribute `data-dojo-props` contains a JavaScript literal that is used to determine whether the element is visible. The value of the `hidden` attribute is determined by sending the length of the `todos` array to custom JavaScript function `emptyConverter`. The function `emptyConverter` returns `true` when the length parameter value is 0, otherwise `false`. The syntax to accomplish hiding an HTML element is fairly complex. On lines 41 to 46 the list element `<ul>` contains the actual list of the to-do items. The list and list item types are defined on lines 42 and 44.

```
32                                      <header id="header">
33                                              <h1>todos</h1>
34                                              <form id="todo-form" data-dojo-attach-event="submit: addTodo">
35                                                      <input id="new-todo" placeholder="What needs to be done?"
data-dojo-type="dojox/mvc/Element" data-dojo-props="_setDisabledAttr: 'domNode', value: at(this, 'newTodo'), disabled:
at(this, 'saving')" autofocus>
36                                              </form>
37                                      </header>
38                                      <section id="main" data-dojo-type="dijit/_WidgetBase" data-dojo-
props="_setHiddenAttr: '', hidden: at(this.get('todos'), 'length').transform(this.emptyConverter)">
39                                              <input id="toggle-all" type="checkbox" data-dojo-type="dojox/mvc
/Element" data-dojo-props="checked: at(this, 'areAllChecked')" data-dojo-attach-event="change: markAll">
40                                              <label for="toggle-all">Mark all as complete</label>
41                                              <ul id="todo-list"
42                                               data-dojo-type="dojox/mvc/WidgetList"
43                                               data-dojo-props="todosWidget: this, children: at(this,
'filteredTodos'), partialRebuild: true, templateString: require('dojo/dom').byId('item-template').innerHTML"
44                                               data-mvc-child-type="todo/widgets/Todo"
45                                               data-mvc-child-mixins="todo/widgets/CSSToggleWidget"
46                                               data-mvc-child-props="todosWidget: this.parent.todosWidget,
_setCompletedAttr: {type: 'classExists'}, _setIsEditingAttr: {type: 'classExists', className: 'editing'}, completed:
at(this.target, 'completed'), isEditing: at(this.target, 'isEditing')">
47                                      </section>
```

Figure 11. Dojo toolkit HTML snippet for to-do list application [21]

Figure 12 illustrates the creation of new to-do list item JavaScript function in `Todos.js` file. On lines 180 to 187 new to-do list item is added asynchronously to the underlying data storage using a combination of `when` and `hitch` functions. The structure indicates that after the function `addStore` finishes successfully, the function given as the second

parameter to the `hitch` function is executed. In case the `addStore` function returns an error response, the function given as the third parameter to the `when` function is executed. The when-hitch structure is architecturally very good. As the creation of a new to-do list item may take some time, it is important that the `addTodo` function is prevented from blocking the running thread by executing the `addStore` function asynchronously. The `hitch` function ensures that the following success or error function is executed in the right context.

```
171                 addTodo: function (event) {
172                         var ret;
173                         var title = lang.trim(this.get('newTodo'));
174                         if (title.length > 0) {
175                                 this.set('saving', true);
176                                 var data = {
177                                         title: title,
178                                         completed: false
179                                 };
180                                 ret = when(this.addStore(data), lang.hitch(this, function () {
181                                         this.get('todos').push(new Stateful(data));
182                                         this.set('newTodo', '');
183                                         this.set('saving', false);
184                                 }), lang.hitch(this, function (e) {
185                                         this.set('saving', false);
186                                         throw e;
187                                 }));
188                         }
189                         event.preventDefault();
190                         return ret;
191                 },
```

Figure 12. Dojo toolkit add to-do list item JavaScript function [21]

Dojo toolkit contains all features required in web application development. Many of the features are structurally well implemented and easy to use. Some features are relatively complex to use and the amount of code required to accomplish simple behaviour can be surprisingly large. The typical way which Dojo is used in HTML is far from being clear and easily understandable.

4.5.3 Ember.js

Ember.js is a fork of another JavaScript framework SproutCore MVC. Ember.js is the youngest of the five compared frameworks started at 2011.

Ember.js is designed to allow developers to concentrate on the application's business logic. This is accomplished by providing lot of abstraction to the common web devel-

opment problems. Ember.js is convention-driven. The conventions apply to many aspects on Ember.js applications. Convention-driven development makes applications appear structurally similar to each other even when they are developed by different parties.

Ember.js provides two-way data binding, templating, components and asynchronous routing. Ember.js component is an isolated view that is similar to AngularJS directive. Ember.js also provides an interesting data module Ember.js Data. The data module integrates tightly with various back-ends allowing easy records retrieval, saving, creation and caching. [11, 8: p.91-93]

Figure 13 illustrates the Ember.js version of to-do list application's HTML for creating new to-do list item and listing existing items. On line 31 `<script>` element defines the use of handlebars templating commonly used with Ember.js. On line 35 a `todo-input` handlebars component is defined for creating new to-do list items. The function to create new to-do list item is defined with the `action` attribute. By looking at HTML it is not clear what the actual input component is. The input component is defined in a separate `todo_input_component.js` file as an extension to `Ember.TextField` component. Notable here is that the handlebars content `{{…}}` will be replaced in the resulting HTML with the actual input component. Simply looking at HTML will not reveal the end result used to generate the layout visible to users.

```
9   <script type="text/x-handlebars" data-template-name="todo-list">
10      {{#if length}}
11          <section id="main">
12              {{#if canToggle}}
13                  {{input type="checkbox" id="toggle-all" checked=allTodos.allAreDone}}
14              {{/if}}
15              <ul id="todo-list">
16                  {{#each}}
17                      <li {{bind-attr class="isCompleted:completed
    isEditing:editing"}}>
18                          {{#if isEditing}}
19                              {{todo-input type="text" class="edit"
    value=bufferedTitle focus-out="doneEditing" insert-newline="doneEditing" escape-press="cancelEditing"}}
20                          {{else}}
21                              {{input type="checkbox" class="toggle"
    checked=isCompleted}}
22                              <label {{action "editTodo" on="doubleClick"}}>
    {{title}}</label>
23                              <button {{action "removeTodo"}} class="destroy">
    </button>
24                          {{/if}}
25                          </li>
26                  {{/each}}
27              </ul>
28          </section>
29      {{/if}}
30  </script>
31  <script type="text/x-handlebars" data-template-name="todos">
32      <section id="todoapp">
33          <header id="header">
34              <h1>todos</h1>
35              {{todo-input id="new-todo" type="text" value=newTitle action="createTodo"
    placeholder="What needs to be done?"}}
36          </header>
37              {{outlet}}
```

Figure 13. Ember.js HTML snippet for to-do list application [21]

The to-do item list is rendered in line 37 replacing the `outlet` handlebar. The to-do list template is defined in lines 9 to 30. The list element `<ul>` on line 15 contains the list items. The `#each` handlebar on line 16 and 26 iterates the to-do list items and creates all elements for each of the item. This type of iterating on the HTML page is very clear and understandable for the reader.

Figure 14 illustrates the creation of new to-do list item JavaScript function in `todos_controller.js` file. On line 11 the title of the new to-do list item is fetched from the Ember.js object structure. Two-way data binding ensures that the object value is the same as presented in the HTML in Figure 13 on line 19. Similarly when the value is reset on line 24 the value displayed in HTML page is reset. The to-do list item is created and added to the storage provided by Ember.js Data module on lines 17 to 21.

```
5   Todos.TodosController = Ember.ArrayController.extend({
6       actions: {
7           createTodo: function () {
8               var title, todo;
9
10              // Get the todo title set by the "New Todo" text field
11              title = this.get('newTitle').trim();
12              if (!title) {
13                  return;
14              }
15
16              // Create the new Todo model
17              todo = this.store.createRecord('todo', {
18                  title: title,
19                  isCompleted: false
20              });
21              todo.save();
22
23              // Clear the "New Todo" text field
24              this.set('newTitle', '');
25          },
```

Figure 14. Ember.js add to-do list item JavaScript function [21]

Ember.js is a very promising JavaScript framework. Many features provided by Ember.js are very useful in the to-do list application development. The code is very clear and understandable. The only bigger issue is the used templating library handlebars. In addition to the replacing handlebars HTML content with the actual HTML elements, handlebars creates massive number of placeholder `<script>` elements making the resulting HTML less readable.

4.5.4 Knockout.js

Knockout.js is a JavaScript framework for creating rich and responsive web applications. Knockout.js supports two-way data binding, HTML templating, MVVM architecture and is easily extensible. Knockout.js does not include routing component but routing can be configured using a third party library. [17]

Figure 15 illustrates the Knockout.js HTML of the to-do list application. The `<input>` element on line 12 is used to add new to-do list items. The element attribute `data-bind` very clearly indicates how the data binding is done and how a new item is added without prior knowledge of Knockout.js. The value of the to-do list item is stored in a variable called `current` and the function triggered when adding a new list item is `add`. The `valueupdate` attribute is used to define the event when the underlying data model is

updated. Notable here is how no content will be changed in the resulting HTML. Viewing the page source with a browser is exactly the same as the Figure 15 content.

The list element `<ul>` on line 17 contains the to-do list items. The `data-bind` element attribute is used to iterate through the list items. The list item element `<li>` on lines 18 to 25 is created for each of the items. The iterating concept is good and the HTML layout clearly indicates which elements will be present in the end result.

```
10                      <header id="header">
11                              <h1>todos</h1>
12                              <input id="new-todo" data-bind="value: current, valueUpdate: 'afterkeydown', enterKey:
    add" placeholder="What needs to be done?" autofocus>
13                      </header>
14                      <section id="main" data-bind="visible: todos().length">
15                              <input id="toggle-all" data-bind="checked: allCompleted" type="checkbox">
16                              <label for="toggle-all">Mark all as complete</label>
17                              <ul id="todo-list" data-bind="foreach: filteredTodos">
18                                      <li data-bind="css: { completed: completed, editing: editing }">
19                                              <div class="view">
20                                                      <input class="toggle" data-bind="checked: completed"
    type="checkbox">
21                                                      <label data-bind="text: title, event: { dblclick: $root.editItem
    }"></label>
22                                                      <button class="destroy" data-bind="click: $root.remove"></button>
23                                              </div>
24                                              <input class="edit" data-bind="value: title, valueUpdate:
    'afterkeydown', enterKey: $root.saveEditing, escapeKey: $root.cancelEditing, selectAndFocus: editing, event: { blur:
    $root.stopEditing }">
25                                      </li>
26                              </ul>
27                      </section>
```

Figure 15. Knockout.js HTML snippet for to-do list application [21]

Figure 16 illustrates the initialization of the to-do list item data model and the creation of a new to-do list item. The to-do list items are stored in a Knockout.js observable array on lines 68 to 70. Knockout.js observables are JavaScript objects that notify subscribers about changes in the data. The creation of a new to-list item on lines 93 to 99 is fairly straightforward. New to-do list item object is created and added to the items array on line 96. The `Todo` object is a simple JavaScript object containing a title and the item state variables. The `bind` function on line 99 sets the `this` context inside the function to match the parent context.

```
65          // our main view model
66          var ViewModel = function (todos) {
67                  // map array of passed in todos to an observableArray of Todo objects
68                  this.todos = ko.observableArray(todos.map(function (todo) {
69                          return new Todo(todo.title, todo.completed);
70                  }));
71
72                  // store the new todo value being entered
73                  this.current = ko.observable();
91
92                  // add a new todo, when enter key is pressed
93                  this.add = function () {
94                          var current = this.current().trim();
95                          if (current) {
96                                  this.todos.push(new Todo(current));
97                                  this.current('');
98                          }
99                  }.bind(this);
```

Figure 16. Knockout.js model and add to-do list item JavaScript functions [21]

Knockout.js development seems fairly logical and straightforward. The HTML layout and JavaScript code is clear and understandable. Knockout.js contains many of the desired JavaScript framework features and the rest can be added using third party libraries. However, the work related in researching additional libraries is undesirable in the NetWiser user interface project scope. Perhaps the biggest drawback with Knockout.js is the missing API documentation. The documentation section in the project web site contains plenty of examples on how a specific feature can be implemented, but there is no list of available JavaScript functions and an explanation on how to use them.

4.5.5 AngularJS

AngularJS's core philosophy is to provide two-way data binding, declarative HTML templating, separation of concerns, dependency injection and extensibility. Declarative HTML templating introduces custom HTML elements and custom HTML element attributes. The goal of declarative HTML templating is to produce concise HTML indicating exactly what it tries to accomplish. Separation of concerns means providing MVVM architectural pattern and separating different parts of application into their own module types such as controllers, services, filters and directives. Dependency injection is a construct to provide service dependencies to application modules. Dependency injection greatly increases the modularity of applications and allows creation of easily plug-

gable independent components. Extensibility allows creation of custom HTML elements and custom HTML element attributes. Extensibility also enables creation of directives to enable the use of other JavaScript library components. [7: p.4-10]

The directives API is sometimes criticized of being too complex. The more advanced concepts of directives such as transclusion, linking functions and scope isolation are somewhat difficult to understand. Another subject for criticism is the silent ignoring of some JavaScript code execution errors making it hard for the developers to track problems. [11]

```
14                              <header id="header">
15                                      <h1>todos</h1>
16                                      <form id="todo-form" ng-submit="addTodo()">
17                                              <input id="new-todo" placeholder="What needs to be done?"
    ng-model="newTodo" autofocus>
18                                      </form>
19                              </header>
20                              <section id="main" ng-show="todos.length" ng-cloak>
21                                      <input id="toggle-all" type="checkbox" ng-model="allChecked"
    ng-click="markAll(allChecked)">
22                                      <label for="toggle-all">Mark all as complete</label>
23                                      <ul id="todo-list">
24                                              <li ng-repeat="todo in todos | filter:statusFilter track by $index"
    ng-class="{completed: todo.completed, editing: todo == editedTodo}">
25                                                      <div class="view">
26                                                              <input class="toggle" type="checkbox"
    ng-model="todo.completed">
27                                                              <label ng-dblclick="editTodo(todo)">{{todo.title}}
    </label>
28                                                              <button class="destroy" ng-click="removeTodo(todo)">
    </button>
29                                                      </div>
30                                                      <form ng-submit="doneEditing(todo)">
31                                                              <input class="edit" ng-trim="false"
    ng-model="todo.title" todo-escape="revertEditing(todo)" ng-blur="doneEditing(todo)" todo-focus="todo == editedTodo">
32                                                      </form>
33                                              </li>
34                                      </ul>
35                              </section>
```

Figure 17. AngularJS HTML snippet for to-do list application [21]

Figure 17 illustrates the to-do list application HTML developed with AngularJS for adding to-do list items and listing the existing items. The adding of new to-do list items is implemented on lines 16 to 18. A `<form>` and `<input>` elements are used to trigger the adding function `addTodo`. The form is used to automatically catch enter key press as it is supported natively in all browsers to send a form. The data model for the new to-do list item is defined with `ng-model` attribute as `newTodo` variable.

The `<ul>` list element on line 23 is used to list the to-do list items. The iteration of the items is defined on lines 24 to 33 in the `<li>` element using the `ng-repeat` element attribute. The list is filtered on the fly by item status using a custom `statusFilter`. Overall

the HTML is clear and it is easy to understand what each part of the page is trying to accomplish.

Figure 18 illustrates the initialization of the to-do list application data model and adding a new to-do list item JavaScript in `todoCtrl.js` file. The to-do list items are initialised on line 12 to a local variable `todos`, to the context `$scope` and the read-only value from the local storage `todoStorage`. The variables are bound to each other. As a result any change to one of them updates the others. A custom watcher is added to the `todos` variable on line 17 to 24. The custom function given as the second parameter to the watcher is triggered when the value of `todos` changes. The function updates the state variables and stores the `todos` variable to the local storage.

```
12  var todos = $scope.todos = todoStorage.get();
13
14  $scope.newTodo = '';
15  $scope.editedTodo = null;
16
17  $scope.$watch('todos', function (newValue, oldValue) {
18          $scope.remainingCount = $filter('filter')(todos, { completed: false }).length;
19          $scope.completedCount = todos.length - $scope.remainingCount;
20          $scope.allChecked = !$scope.remainingCount;
21          if (newValue !== oldValue) { // This prevents unneeded calls to the local storage
22                  todoStorage.put(todos);
23          }
24  }, true);
25
34
35  $scope.addTodo = function () {
36          var newTodo = $scope.newTodo.trim();
37          if (!newTodo.length) {
38                  return;
39          }
40
41          todos.push({
42                  title: newTodo,
43                  completed: false
44          });
45
46          $scope.newTodo = '';
47  };
```

Figure 18. AngularJS model and add to-do list item JavaScript functions [21]

The to-do list item add function starts on line 35. The function is very simple. Noticeable here is that the created to-do list item object is a standard JavaScript object rather than AngularJS specific data object implementation.

AngularJS provides a very comprehensive set of web application development tools. The declarative style of producing HTML is clear and quite easy to understand. The

declarative style allows developers to create HTML content with relatively small amount of HTML markup. AngularJS does not seem to have any bigger problems in relation to the NetWiser user interface project. The only notable drawback is the lack of Internet Explorer 8 support after AngularJS version 1.2. AngularJS 1.3 was released in October 2014 after which version 1.2 gets only bug fix releases.

4.5.6 Framework Popularity

One important factor to take into account in selecting JavaScript framework is the overall interest around the framework. Interest includes the number of developers participating in the development of the framework, content available in various resources and the amount of activity in the social media. All of these measures are individually only indicative on how popular a particular framework is. However, the combination of different metrics gives an idea whether a framework is popular compared to another framework. Popularity and overall activity around a framework help in determining the future of the framework. The selected framework should be actively developed for many years to come.

Figure 19 illustrates various metrics on each of the frameworks. The first item in the figure is the number of third party modules created for the framework. Many frameworks provide some way of extending the framework and a web site where the modules can be shared with the community. AngularJS has the largest module base.

All of the frameworks are hosted in GitHub. The second item in the list is the number of GitHub contributors which is the number of people that have modified the software. The number of contributors is highest in AngularJS and Ember.js. GitHub stars indicate the interest among GitHub user's towards a specific repository. It is also an indication of appreciation towards the repository. AngularJS has the highest star count.

Stack Overflow (http://stackoverflow.com) is a widely used web site for programmers for asking and answering programming related questions. The fourth item in Figure 19 is the number of questions in Stack Overflow tagged for a specific framework. The number of questions indicates an overall interest towards a framework among programmers. Working with a programming task raises questions on how a specific feature can be implemented. Stack Overflow contains plenty of these type of questions and answers to the questions. As such the questions and answers become a very use-

ful resource for developers. AngularJS has the highest question count in Stack Overflow.

YouTube results is the sixth item in Figure 19. YouTube contains teaching and presentation videos for frameworks among other material. AngularJS has the most content available in YouTube. The seventh item in the figure is the number of Twitter tweets in a thirty-day period from January 3rd to February 4th 2015. In this time period AngularJS has the most activity in Twitter.

The last item in Figure 19 is the number of Chrome extension users. A Chrome extension here stands for a development extension created specifically for the framework. AngularJS has the largest user base for their Chrome extension Batarang.

Framework	AngularJS	Backbone.js	Dojo toolkit	Ember.js	Knockout
3rd party modules [1)]	1 192	249	N/A	552	61 [2)]
GitHub contributors	450	246	64 [3)]	406	49
GitHub stars	34 862	20 590	427 [3)]	12 597	5 991
Stack Overflow questions	75 713	17 295	7 559	13 758	12 821
YouTube results	102 000	15 400	2 580	9 590	9 770
Tweets in the last 30 days [4)]	29 772	401	123	512	456
Chrome extension users [5)]	214 273	10 160	N/A	51 338	18 306

[1)] Web sites: ngmodules.org, backplug.io, emberaddons.com, github.com/knockout/knockout/wiki/Plugins

[2)] There is no web site to track Knockout extensions, only a known plugins list in Knockout wiki

[3)] GitHub main project dojo / dojo

[4)] According to topsy.com social analytics

[5)] AngularJS Batarang, Backbone Debugger, Ember Inspector, Knockoutjs context debugger

Data collected Feb 3rd 2015

Figure 19. Key JavaScript framework popularity

Figure 20 illustrates Google Trends activity for each of the frameworks from January 2005 to January 2015 [22]. Google Trends interest is based on the amount of searches conducted with Google's search engine. In October 2012 Backbone.js was the most trending framework and since then AngularJS has significantly surpassed the others.

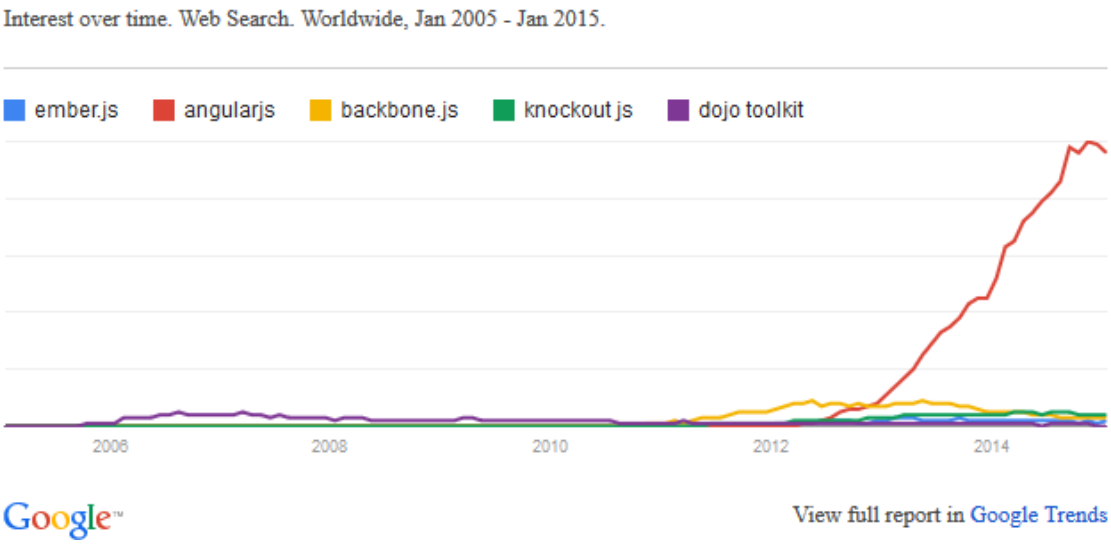

Figure 20. Google Trends on key JavaScript frameworks

Based on these metrics AngularJS is clearly the most popular framework at the moment and in the near past. The second most popular framework is a lot harder to determine based on the metrics. In some metrics the number two is Backbone.js and in some metrics it is Ember.js.

4.6 Conclusion

The framework selected for NetWiser user interface development was AngularJS. NetWiser user interface could be developed with any of the final five frameworks. AngularJS was chosen because it is the best fit for the project. In overall comparison between the five frameworks AngularJS provided a solution that results in the highest number of benefits and the least amount of issues during development. Some of the key framework shortcomings are listed in the following paragraphs affecting the selection of AngularJS.

Backbone.js does not provide some of the key web application development features such as two-way data binding. Backbone.js forces the development team to choose additional libraries to supplement the missing features. Web applications can be developed without two-way data binding but it is a really nice feature to have.

HTML written using Dojo toolkit looks very complicated and messy. It is the common practice of including JavaScript functions inside HTML element attributes that makes

the interpretation of Dojo toolkit HTML challenging. Some simple web application behaviour is fairly complicated to implement with Dojo toolkit. It is not that something cannot be done using Dojo toolkit. The framework does not provide many features directly and as a result the feature has to be programmed by the developer.

The most notable problem with Ember.js is the commonly used templating system handlebars. The readability of the produced HTML suffers with handlebars as the handle {{…}} content gets replaced with something else. In addition, every handle adds a `<script>` element to the resulting HTML making it difficult to find the content that provides the application functionality.

Some key web application development features are missing from Knockout.js. Knockout.js does not provide URL routing making it necessary to turn to third party libraries. Knockout.js documentation does not provide an API documentation that lists all the available JavaScript functions and their description. The lack of API documentation is surprising given the flexible nature of JavaScript language. For example a JavaScript function can be called with various numbers of parameters making it challenging to understand the function's full nature by simply looking at example code.

AngularJS provides a better solution to all of the previous shortcomings in the other frameworks. AngularJS does not seem to be missing any key features that are implemented in the other frameworks. AngularJS provides the most comprehensive solution for web application development. As AngularJS is the most popular framework, the choice in favour of AngularJS is not a difficult one.

5 NetWiser Web User Interface Development Project

The NetWiser web user interface development project was the case project for the thesis. The project is the first phase in a bigger architectural change planned for the NetWiser product. The goal of the project was to solve detected technical and design issues. All architectural and design decisions made during the project should consider the effects on the overall product architecture. Planned future changes to the product backend will affect the user interface communication layer. Following the best practices such as application modular structure enables easier communication layer modification in the future.

This section covers the NetWiser user interface project phases. The beginning of the section introduces the project in general and describes the selected tools and project structure. The latter part of the section covers a number of application portions that demonstrate how AngularJS is employed in implementing various application features. The demonstration also includes application separation of concerns into AngularJS modules such as services, controllers and directives.

5.1 Project Scope and Architectural Requirements

The development project includes the technology switch from Java applets to JavaScript AngularJS framework. The same application logic is transferred from one technology to another. The multi-window user interface layout is changed to present everything on a single page using user interface layers instead of additional windows. User interface components where user experience is not at optimal level were replaced with components commonly used in current user interface development. The planned redevelopment of the overall layout and component placing are excluded from the project scope. The content provided by the dynamic HTML from the data backend is left untouched. Dynamic HTML content is present in the data view frames and navigation.

Project architectural requirements include some industry practices that are commonly used in user interface projects. The architectural requirements originate from the overall product architecture change plan. The first requirement is user interface automatic testability. In practice this means an employment of an automated test framework or

tool designed for user interface testing. The second requirement is the technology modernization into one of the leading JavaScript frameworks. AngularJS, as the research suggests, can be characterized as one of the leading JavaScript frameworks. A third requirement is the use of a continuous integration environment. Continuous integration stands for automated software build process and testing after every change to the versioning system. In addition to the automated user interface tests, automated unit tests are included in the project scope to add more depth to testing.

5.2 Project Setup and Tools

Project setup follows the best practices of the industry. The setup includes tools for all of the project's needs from programming, testing and distribution. JavaScript based projects have a considerable number of tools available. Tools are mostly selected based on recommendations on AngularJS development. Some tools such as source code editor is chosen based on personal preference. Setting up the project in a good manner and selecting best possible tools allows the developers to concentrate on developing the application features. Later when the application enters to the maintenance mode, maintenance work is easier due to the clear project structure and available tools.

Eclipse IDE – Integrated Development Environment

Eclipse IDE has become known for its Java development capabilities. Eclipse IDE also supports other programming languages including JavaScript [19]. The IDE is chosen as the programming tool and text editor based on developer preference. AngularJS Eclipse plug-in provides an HTML editor adding support for AngularJS specific expressions and directives. The plug-in is installed to Eclipse IDE using Eclipse Marketplace feature.

Node.js

Node.js is utilised on the project for two reasons. Firstly, Node.js provides the npm command-line tool for JavaScript package management. The npm package manager is commonly used in JavaScript projects. Other JavaScript tools can be easily installed with the package manager using a single command. Secondly, Node.js provides an integrated web server. The web server is used as the development web server and as an end-to-end test web server. The web server can be started with one command and

a simple setup requires minimal configuration. Project tool dependencies are listed in `package.json` file. The dependencies are installed using the package manager with a command:

```
$ npm install
```

Dependency Management Tools Bower and Bower Installer

The project dependency management tool is called Bower. A dependency management tool is used to manage project's third party library dependencies. Bower downloads and installs entire library repositories to the local hard drive. Bower Installer is utilised to select resources from the downloaded repositories to be included in the project. Some libraries do not support Bower requiring custom configuration with Bower Installer. Resources are installed in a configured directory under the project structure. Bower Installer also allows exclusion of unnecessary development libraries from the project distribution package. Bower and Bower Installer configuration is stored in `bower.json` file. Third party libraries are downloaded and the dependencies installed with the following commands:

```
$ bower install
$ bower-installer
```

Task Tool Grunt

A software project distribution package creation is a set of repetitive tasks and scripts executed in the same manner every single time. Commonly, repetitive tasks are run with a task runner in a software project. Grunt is the automated task runner selected for the project. The tasks include JavaScript minification, unit testing and distribution package creation among others. The Grunt configuration is stored in a project file called `Gruntfile.js`. The default Grunt task tests and creates the application distribution package. The task can be executed with a command:

```
$ grunt default
```

Version Control with Git and SVN

Version control systems have two purposes in the project development. The first purpose is the dependency management tool Bower requirement. Bower uses Git to download third party library repositories from the Internet. The second purpose is the project version control. The project is version controlled in an SVN repository.

Test Tools Jasmine, Karma and Protractor

The project is automatically tested using unit tests and end-to-end tests. Unit test tests a small unit of the JavaScript source code. End-to-end test tests an application feature using a real web browser. Jasmine is the test framework used for creating unit tests. Karma is the test runner for the unit tests. Karma does not require a real browser to run the unit tests, instead a headless JavaScript running environment PhantomJS is used. Headless running environment is useful in avoiding unnecessary browser start-up when running the unit tests as part of the distribution package creation Grunt task. Protractor is the end-to-end test framework developed for AngularJS. Protractor utilises Selenium Web Driver to control the web browser used in testing.

Continuous Integration with Jenkins

Continuous integration tool Jenkins is automatic project build tool. Build steps include automated tests and creation of the project distribution package. The build job is executed after every source code change committed to the versioning system. The tool also automatically deploys a new version of the application to the test server after a successful project build job. Continuous integration ensures the integrity of the software package throughout the development phase.

Yeoman

Yeoman is a tool for creating a project template. One of the Yeoman supported project templates is AngularJS template. The template includes necessary project build and dependency files like `package.json`, `Gruntfile.js` and `bower.json`. The template also includes a very basic AngularJS application that can be used as a starting point for any AngularJS application. Yeoman greatly reduces menial work involved in starting up an

application project. The default Yeoman AngularJS project structure is generated with command:

```
$ yo angular
```

Project structure

The project directory structure is based on Yeoman generated structure and recommendations from the literature and online guides. The directory structure for AngularJS applications has not been standardized and a common practise that applies to most use cases has not been established. However, a suggestion for the recommended AngularJS application directory and file structure exists [20].

Figure 21 illustrates the project root and app directory structures. The application source code and resource files reside in the `app` directory. Project third party library dependency repositories are located in the `bower_components` directory. The `dist` directory contains the application distribution package created with Grunt. Node.js based tool dependencies are located in the `node_modules` directory. Project scripts such as development server start-up script and end-to-end test script are located in the `scripts` directory. Test cases and test configurations for both unit and end-to-end tests reside in the `test` directory. Test results are generated to `test_out` directory. The root directory also contains individual configuration files for Node.js, Grunt and Bower.

Figure 21. Project root and app directory structures

The `app` directory corresponds very closely to the distribution directory structure. For this reason the development web server can be started using the `app` directory as the base directory. The `css` directory contains cascaded style sheets for the application and third party libraries. Application image resources reside in the `images` directory. The application and third party JavaScript source code is located in the `js` directory. The directory `nw-web` contains necessary HTML files from the old user interface. The old HTML files are used to provide data backend static content like login and error pages. Development web server backend mimicking content is located in the `test_content` directory. HTML files reside in the `app` directory root.

5.3 Product Architecture and User Interface Design

New NetWiser user interface architecture follows the original architecture as much as possible. Figure 22 illustrates the new NetWiser component architecture. Comparison between the old architecture in Figure 1 on page 5 and the new architecture in Figure 22 reveals that only one component has changed. The Java applet component has been replaced with JavaScript framework component on the user interface. Parts of the user interface that were provided by the Java applet are replaced with HTML.

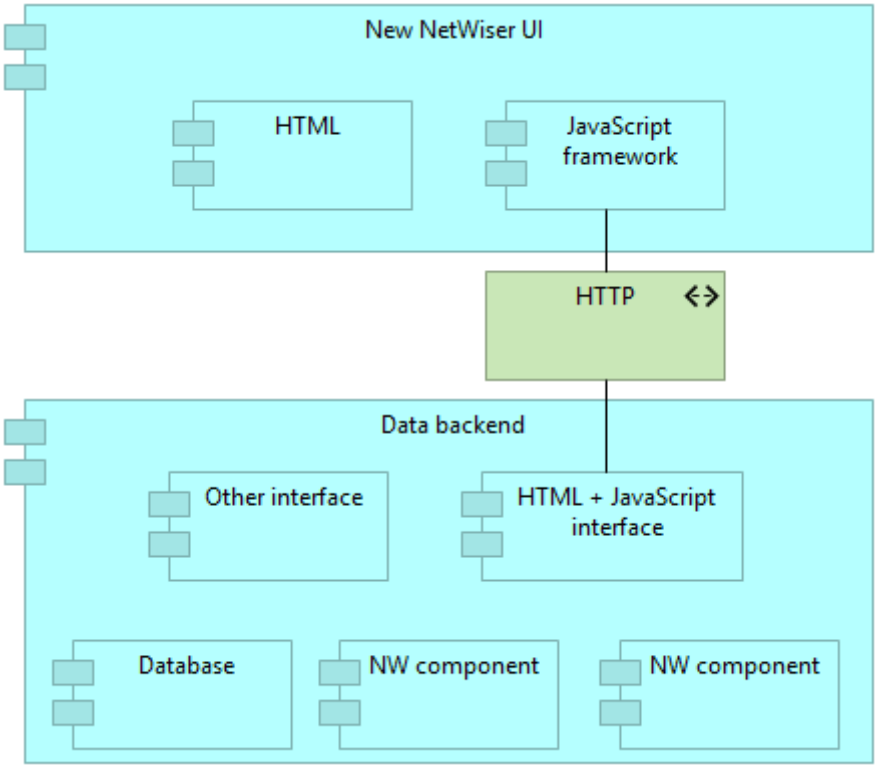

Figure 22. New NetWiser component architecture

The new NetWiser user interface design is not changed much from the original design. The main view overall layout remains exactly the same as the original. The search parameter frame is at the top, the data view frames remain at the centre and the navigation at the bottom. The separate search parameter selection window is brought on top of the main view as a modal layer. The different search parameter types are divided to separate sections using a tab component.

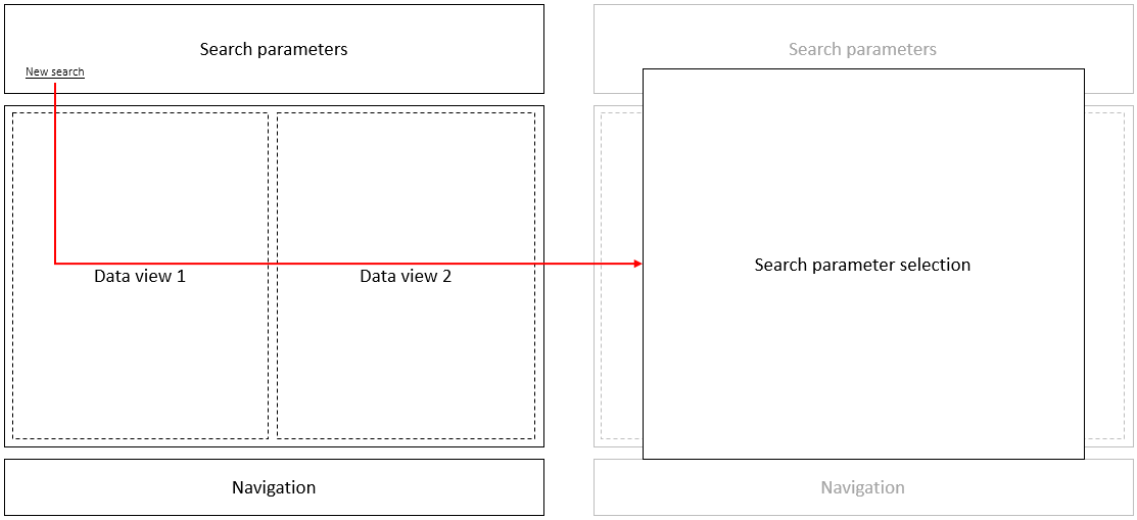

Figure 23. New NetWiser user interface design

The layout designed for this project phase is a temporary solution between the old layout and the upcoming layout. The layout will be redesigned in a later project phase. The later project phase is not included in the thesis scope.

5.4 User Interface HTML Structure

The main entry point to the NetWiser user interface application is the `index.html` file. Resources like JavaScript files and cascaded style sheets are defined in the `index.html` file. NetWiser user interface visible content selection is also included in the `index.html` file as illustrated in Figure 24. The authorization controller `authCtrl` is defined on line 20 stating that the authorization controller context is bound to the `body` section of the `index.html` file. The content selection is implemented on lines 22 to 26 and on lines 28 to 33. The visible content is selected based on the authorization controller `loggedIn` flag variable. The flag variable indicates whether the user has logged in to the application. The `login.html` page content declared on line 23 is hidden when the user has not logged in as stated on line 24 using the `ng-hide` attribute. The

`ogd_search.html` page visibility condition uses another kind of approach compared to the `login.html` visibility condition. The usage of `ng-if` and `ng-include` on lines 29 and 30 ensure that the search controller declared on line 31 is instantiated only after the user has logged in. The search page is shown when the user has logged in.

```
20 <body ng-controller="AuthCtrl as authCtrl">
21
22     <div class="content"
23         ng-include="'login.html'"
24         ng-hide="authCtrl.loggedIn"
25         ng-cloak>
26     </div>
27
28     <div id="ogd-search" class="content"
29         ng-if="authCtrl.loggedIn"
30         ng-include="'ogd_search.html'"
31         ng-controller="SearchCtrl as searchCtrl"
32         ng-cloak>
33     </div>
```

Figure 24. NetWiser user interface HTML content selection

The login page is illustrated in Figure 25. The authorization controller is declared in the `index.html` file enabling the controller in the `login.html` page context as well. The login functionality is provided using an HTML `form` element on line 2. The AngularJS form submit is defined with the `ng-submit` attribute. Submitting the form triggers the `login` function in the authorization controller. The user name and password input elements are defined on lines 6 and 10. Both elements are bound to a variable in the authorization controller using the `ng-model` attribute. The login type selection is implemented on lines 13 to 16. The `ng-repeat` attribute iterates over the login types defined in the authorization controller on line 13. Each login type has a key and a description. A radio button and a label is generated for each login type on lines 14 and 15 using the login type key as the radio button value and the description as the label value.

```
<form class="login" ng-submit="authCtrl.login()" role="form">
    <h2>NetWiser login</h2>
    <div>
        <label class="login">User id:</label>
        <input class="login" ng-model="authCtrl.auth.username" autofocus/>
    </div>
    <div>
        <label class="login">Password:</label>
        <input class="login" ng-model="authCtrl.auth.password" type="password"/>
    </div>
    <div id="login-error" class="login-error" ng-bind="authCtrl.loginError"></div>
    <div ng-repeat="type in authCtrl.authTypes">
        <input ng-model="authCtrl.auth.type" ng-value="type.key" type="radio"/>
        <label ng-bind="type.description"></label>
    </div>
    <div><button id="login-button" type="submit">Log in</button></div>
</form>
```

Figure 25. NetWiser login page HTML

The search page contains a section where pre-defined search profiles can be selected using a drop down selection component. The search profile selection HTML is illustrated in Figure 26. The selection is implemented using an HTML select element. The selection value is stored and retrieved from the search controller variable `selectedSearchProfileName` using the AngularJS two-way data binding mechanism. The selection options are generated using the AngularJS specific expression as the `ng-options` attribute value. The list of selection options is retrieved from the search controller variable `searchProfileNames`. Once the user selects another value in the drop down menu, function `profileChanged` defined with the `ng-change` attribute is triggered.

```
        <select ng-model="searchCtrl.selectedSearchProfileName"
                ng-options="s for s in searchCtrl.searchProfileNames"
                ng-change="searchCtrl.profileChanged()"
                ng-disabled="searchCtrl.updating">
        </select>
```

Figure 26. Search profile selection HTML

5.5 NetWiser Services

NetWiser user interface services implement the AngularJS service concept. In AngularJS services are individual application modules that provide a service API (Application Programming Interface) and can be replaced with another implementation of the service. Services are wired together and to other components using another AngularJS concept called dependency injection [7: p.69-74]. NetWiser provides two service APIs:

one for authentication and one for search. Both APIs communicate with the data backend.

NetWiser authentication API consists of four functions. Two of the functions handle the authentication actions login and logout. The other two functions implement the authentication verification functionalities. The verification functions are used to check the user login status based on the server response. The authentication service login function is illustrated in Figure 27. The `AuthService` module is created on line 6 using the AngularJS `factory` function. The `factory` function is one of the three available AngularJS functions for creating service modules [7: p.69]. The code on line 6 states that the `AuthService` is dependent on AngularJS core service `$http`. AngularJS uses its dependency injection mechanism to pass the `$http` service instance to the `AuthService`. The `AuthService` API variable with a structure is created on line 9. The APIs typically consist of functions and variables. The `AuthService` API reference is returned on line 42 as a variable providing the authentication functions. The `AuthService` API can be used in other parts of the application using similar dependency injection as demonstrated on line 6.

```
5  angular.module('nw.services', [])
6    .factory('AuthService', function($http) {
7
8      // Authentication service API
9      var authServiceApi = {
10         login: function(auth) {
11           console.log('Logging in user: ' + auth.username);
12           return $http({
13             method: 'post',
14             cache: false,
15             url: '/tflex-bin/auth',
16             data: 'MODE=logon&userid=' + auth.username + '&password=' + auth.password + '&CONN_TYPE=' + auth.type,
17             headers: {
18               'content-type': 'application/x-www-form-urlencoded'
19             }
20           });
21         },
41     };
42     return authServiceApi;
43   })
```

Figure 27. Authentication service login function

The figure illustrates one of the four API functions called `login`. The `login` function employs the `$http` service to communicate with the NetWiser data backend `auth` service interface on lines 12 to 19. The `login` function returns a promise object created by the `$http` service call. AngularJS promise objects are used in asynchronous operations to achieve non-blocking long lasting operations, response handler callback functions for successful and failed operations and chaining of asynchronous operations in a synchronous manner [7: p.90-92].

5.6 NetWiser Controllers

NetWiser user interface controllers implement the AngularJS controller concept. AngularJS controllers bind the user interface to the data model and vice versa. Controllers implement the application business logic and handle user interactions [7: p.17]. NetWiser user interface consists of two controllers: authorization controller `AuthCtrl` and search controller `SearchCtrl`. Authorization controller handles user authentication related interactions login and logout. Search controller handles search related user interactions and population of search selection menus with appropriate data.

Authorization controller provides two functions for user interactions: login and logout. The authentication controller initialization and login function are illustrated in Figure 28. The authentication controller is initialised on lines 5 to 6 by first declaring the NetWiser controllers module on line 5 and by creating the authentication controller on line 6. The authentication controller uses dependency injection to state the controller's dependency to the authentication service `AuthService` on line 6.

```
5  angular.module('nw.controllers', [])
6    .controller('AuthCtrl', function(AuthService) {
7      var self = this;
8      var httpErrorHandler = function(error) {
9        console.error('Error in backend communication: ' + error.statusText);
10     };
11
23
24     self.login = function() {
25       AuthService.login(self.auth).then(function(response) {
26         if(AuthService.isAuthenticated(response)) {
27           self.loggedIn = true;
28           self.loginError = null;
29           self.auth.username = null;
30           self.auth.password = null;
31           self.auth.type = 'NW';
32         } else {
33           self.loginError = 'Invalid user id or password';
34         }
35       }, httpErrorHandler);
36     };
```

Figure 28. Login controller login function

The login function is bound to the login form submit action in `login.html` file illustrated in Figure 25 on line 2. The authorization controller login function is illustrated in Figure 28 on lines 24 to 36. Once the login action has been triggered by the user, the controller login function calls the authentication service login function with the authentication

information `auth` as a parameter. The authentication login function returns a promise object synchronously while the asynchronous backend operation may not yet be completed. The promise object `then` function takes success and error callback functions as parameters. As the login operation in authentication service is asynchronous, the success or error callback function is called only after the operation completes. The success callback function on line 26 checks whether the authentication was successful. Based on the authentication status the user is logged in or an error message is displayed. On successful login the user authentication variables are reset. Once the variable `loggedIn` changes to `true` on line 27, AngularJS two-way data binding ensures that the login page becomes hidden and the search page becomes visible as defined in `index.html` file illustrated in Figure 24.

5.7 NetWiser Directives

NetWiser user interface directives implement the AngularJS directive concept. AngularJS directives are used to modify component behaviour and create reusable components [7: p.169-170]. NetWiser employs the directives concept to add user interface tabs component and to include jQuery UI date picker component. The date picker directive is illustrated in Figure 29. The directive variables `restrict`, `require` and `link` configure the directive and provide the directive behaviour.

The directive variable `restrict` indicates which type of HTML components can trigger the directive. The date picker behaviour directive is restricted to HTML element attribute name using value `'A'` on line 58. Other possible values are `'E'` for HTML element name and `'C'` for HTML class name. The restriction value can also be combined for example by providing value `'AEC'`. The `require` variable lists the directive's controller dependencies. In the date picker directive on line 59 the `ngModel` is indicated as a dependency. The dependency means that any HTML element utilising the `jq-date-picker` attribute must also utilise the `ng-model` attribute.

```
56  .directive('jqDatePicker', function () {
57    return {
58      restrict: 'A',
59      require: 'ngModel',
60      link: function($scope, $element, $attrs, ngModelCtrl) {
61        $element.datetimepicker( {
62          dateFormat: 'd.m.yy',
63          timeFormat: 'H:mm',
64          firstDay: 1,
65          onSelect:function (date) {
66            $scope.$apply(function () {
67              ngModelCtrl.$setViewValue(date);
68            });
69          }
70        });
71      }
72    };
73  })
```

Figure 29. jQuery UI date picker directive

The directive variable `link` defines a function that is used to modify the DOM tree. The link function parameters are `$scope` referring to AngularJS scope, `$element` referring to the HTML element triggering the directive and `$attrs` referring to the triggering HTML element attributes. In addition the required controllers are passed as additional parameters. In the date picker directive the model controller is passed as the fourth parameter on line 60. The jQuery UI date picker is extended with a jQuery UI time picker add-on. The add-on adds time fields to the jQuery UI date picker. The date time picker is initialised on line 61 adding the component to the directive triggering HTML element. The first day of the week, date and time formats are configured with variables `dateFormat`, `timeFormat` and `firstDay`. The `onSelect` variable defines a function triggered when the date picker is selected by the user. The function sets the date value to the date picker component using the controller's `$setViewValue` function on line 67. This method in association with scope's `$apply` function is a typical implementation of extending the AngularJS two-way data binding to external components.

5.8 User Interface Improvements

Changes to NetWiser user interface layout and component placing are excluded from the project scope. However, some user interface components are replaced with components providing better user experience. Figure 30 illustrates the date and time selector components. The old selector is located on the left part of the figure and the new selector on the right. The old selector consists of five individual selectors for each time

component including day, month, year, hour and minute. The new selector provides more intuitive selection using a calendar type selector for date and sliders for time.

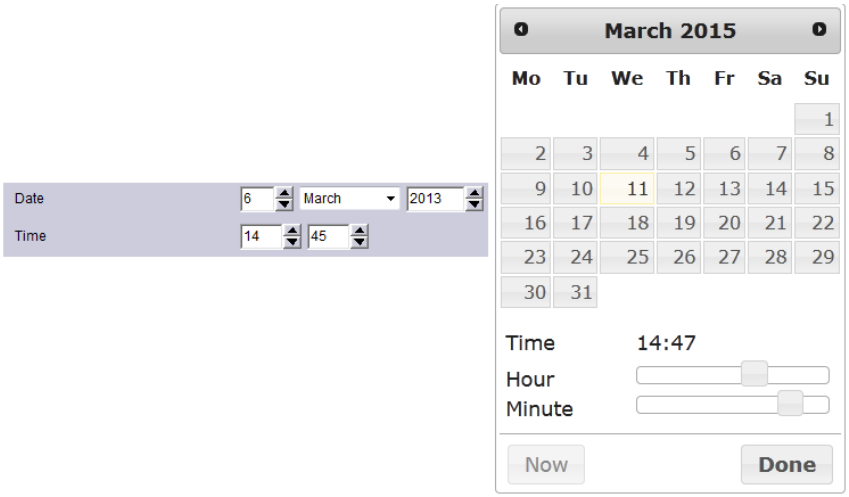

Figure 30. Date and time selector improvements

Field selector components are illustrated in Figure 31. The old selector is located at the left part of the figure and the new selector is located at the right. The selector is used to select fields displayed on the search results table. Available fields are listed in the left menu and the selected fields are listed in the right menu in both selectors.

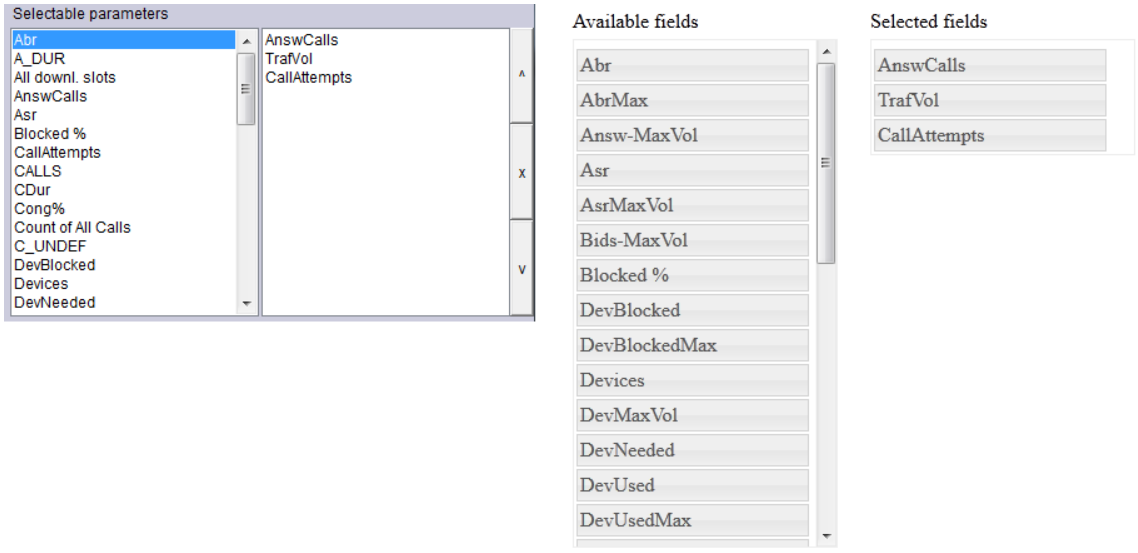

Figure 31. Field selection improvements

The selection in the old menu is accomplished by clicking a menu item on the left menu. The item is copied at the bottom of the right menu. Similarly the new selector

allows field selection by double clicking an item and in addition the item can be dragged to the desired location. The selected items are arranged in the old menu using the up and down buttons at the right and removed by clicking the `X` button. The selected items are arranged and removed in the new menu by dragging the component to the desired location. The removal is also achieved by double clicking the item in the right menu.

5.9 Unit, End-to-end and Manual Testing

Testing is a very important part of application development. NetWiser user interface testing is included in the development process by utilising automated tests rather than as a phase after the coding has been completed. Automated tests consist of unit tests and end-to-end tests. Unit test tests a small part of the application JavaScript code. End-to-end test tests an application feature using a real web browser. Unit tests use the Jasmine test framework and are executed using the Karma test runner. Protractor is the test framework used with the end-to-end tests. Jasmine is selected as the test framework for Protractor to use internally. Manual tests are conducted using an actual web browser by a user. Manual tests are conducted to catch situations where the application logic and the automated tests are both wrong as a result of a developer misunderstanding.

An example unit test for the data service search profile names is illustrated in Figure 32. The `describe` function on line 80 is used to create a test suite for data services. The `beforeEach` functions on lines 81 to 82 initialise AngularJS modules required by tests. All `beforeEach` functions are executed before each test. The `httpBackendMock` and `dataService` variables are initialised for each test using the AngularJS dependency injection on lines 87 to 90. The `$httpBackend` creates a fake HTTP interface designed for AngularJS test purposes. The `afterEach` function on lines 92 to 95 verifies that all HTTP expectations and requests have been fulfilled. Similarly to `beforeEach` function execution, all `afterEach` functions are executed after each test. Test cases are called specs and they are defined using an `it` function. A spec to load search profile names using the data service is located on lines 97 to 109. On line 99 an expectation for an HTTP GET method is declared towards the fake HTTP backend with a static response. When the data service executes an HTTP GET method with the specified URL, the service gets the specified response defined with `SEARCH_PROFILES_RAW` variable. Search

profiles are loaded and parsed using the data service on lines 101 to 104. As the `searchProfiles` function returns a promise object, the handling of the response is conducted using the promise object's `then` function. The response parsing on line 103 is executed after the HTTP backend function `flush` on line 107 has been called. The expectations on lines 105 and 108 compare the `searchProfiles` variable first to and empty array and then to an array defined with `SEARCH_PROFILES` variable.

```
80  describe('data services', function() {
81    beforeEach(module('nw.services'));
82    beforeEach(module('dateParser'));
83
84    var httpBackendMock;
85    var dataService;
86
87    beforeEach(inject(function($httpBackend, DataService){
88      httpBackendMock = $httpBackend;
89      dataService = DataService;
90    }));
91
92    afterEach(function() {
93      httpBackendMock.verifyNoOutstandingExpectation();
94      httpBackendMock.verifyNoOutstandingRequest();
95    });
96
97    it('should load search profile names', function() {
98      var url = '/tflex-bin/genmenu?MODE=RAW&PROGRAM=OGD&MSG=getprofiles';
99      httpBackendMock.expectGET(url).respond(SEARCH_PROFILES_RAW);
100
101     var searchProfiles = [];
102     dataService.searchProfiles().then(function(response) {
103       searchProfiles = dataService.parseSearchProfiles(response.data);
104     });
105     expect(searchProfiles).toEqual([]);
106
107     httpBackendMock.flush();
108     expect(searchProfiles).toEqual(SEARCH_PROFILES);
109   });
```

Figure 32 Data service search profile names loading unit test

An example end-to-end test is illustrated in Figure 33. The test suite is declared with the Jasmine `describe` function similarly to unit tests on line 7. A function executed before each test case on lines 9 to 12 disables the browser synchronization and loads the front page of the application from the test server. A test case testing failed login attempt is on lines 14 to 22. First, the initial state of the page is evaluated by checking that the HTML element with id `ogd-search` is not present and HTML element with id `login-error` contains no text. Then, invalid username and password are entered to the input

fields and the login button is clicked. After the login attempt, the page state is checked not to contain the search element and to contain some text in the login error element.

```
7   describe('auth', function() {
8
9     beforeEach(function() {
10      browser.ignoreSynchronization = true;
11      browser.get('http://localhost:8100/');
12    });
13
14    it('should fail login', function() {
15      expect(element(by.id('ogd-search')).isPresent()).toBe(false);
16      expect(element(by.id('login-error')).getText()).toEqual('');
17      element(by.model('authCtrl.auth.username')).sendKeys('system');
18      element(by.model('authCtrl.auth.password')).sendKeys('abc');
19      element(by.id('login-button')).click();
20      expect(element(by.id('ogd-search')).isPresent()).toBe(false);
21      expect(element(by.id('login-error')).getText()).toNotEqual('');
22    });
```

Figure 33. Failed user login end-to-end test

Automated tests are run after every committed code change to the versioning system by the continuous integration system. Continuous test execution ensures that errors are detected as early as possible. All automated tests are executed several times every day. Continuous integration system does not allow delivering the software package to a test server before all automated tests pass. This practice encourages developers to fix detected errors immediately. Manual user tests are executed periodically as the features are developed. NetWiser user interface features are duplicated to the new user interface allowing manual tester to compare both interfaces side by side. At the end of the project development phase all automated unit and end-to-end tests were passing. Manual user testing no longer found any functional differences between the old and the new user interfaces.

6 Summary and Conclusions

The purpose of the thesis was to develop a new user interface for the NetWiser product. To reach the objective, a research on JavaScript frameworks was conducted and a suitable JavaScript framework was selected. The user interface project was carried out by following the best practices of the selected JavaScript framework AngularJS.

6.1 Selection of JavaScript Framework

The first part of the research was to select a suitable JavaScript framework to implement a new NetWiser user interface. The selection was conducted by examining the features of 48 JavaScript frameworks and libraries. The selection was divided into three phases where each phase eliminated a number of frameworks. The first and second phase eliminated frameworks not fulfilling basic and technical requirements. The third phase investigated the remaining five frameworks by examining the same example application developed with each of the frameworks and by comparing framework popularity. After a careful examination, AngularJS was chosen as the most suitable framework for the project.

The selection of AngularJS framework was a success. The framework introduces multiple useful features that elevate the development process to a higher level compared to development work conducted without a framework. In other words, developers can concentrate more on the application business logic and less on the menial development work such as keeping the user interface and JavaScript model objects in sync. One of the best framework features is the modular application structure. Modular application structure enables reusable component creation and ensures maintainable project structure.

Development work very often involves searching the literature or the Internet for how a specific feature can be implemented with the development framework and programming language used. The quality of the search results depend highly on the size of the community using the framework. A bigger community increases the probability of the search topic having already been discussed somewhere in the Internet. A bigger community also encourages publishers to commission books on the subject. During this

project it became apparent that the AngularJS community is big enough to support professional development work. The practical part of the project involved finding an answer to dozens or hundreds of small AngularJS related problems. All questions were answered in the literature or in the Internet. Overall, using AngularJS for development was a positive experience.

Advanced user interface components such as tabs, date pickers and accordions are not part of the AngularJS framework. In retrospect, user interface component coverage should have been included in the study. However, the AngularJS community provides a vast number of third party modules including advanced user interface components. It is also relatively easy to implement AngularJS directives to enable the use of jQuery UI components or other existing JavaScript components.

6.2 NetWiser User Interface

The goal of the NetWiser user interface project was to replace the current Java applet technology with a JavaScript framework and to replace some of the user interface components to improve user experience. Complete user interface layout redesign was excluded from the scope of the thesis project. The current study is part of a bigger project to further develop various parts of the NetWiser product. The architectural and design decisions of the thesis project were to support the whole product development.

The implication of the technology change was to completely redevelop a large part of the NetWiser user interface application. The technology change successfully removed the dependency on Java Runtime on user workstations. The change was carried out by replacing the Java applet application component with a new application component developed with the AngularJS JavaScript framework. Other product components and communication interfaces were not touched. The best development practices were followed, including project tools and implementation. The project tools include integrated development environment, dependency management, testing, version control and continuous integration tools. The implementation followed AngularJS guidelines found in literature and online documentation. Some parts of the AngularJS development, such as project directory and file structure, do not currently have one standardized practice.

Some individual user interface components were replaced by components that conform to the current user interface usability standards. As an example a date and time selector component using multiple individual selector components was replaced with a modern date time picker component with a calendar appearance. The old and new date time components are illustrated in Figure 30 on page 53. User interface component replacements resulted in a better user experience. The positive result is based on a consensus between the project team and a customer.

The application operation was successfully verified using automated unit tests, automated end-to-end tests and manual user tests. As the functionality of the application remained identical to the old user interface, manual user tests could be executed by comparing the functionality of the applications side by side. A demo session with a customer resulted in a few user interface improvement ideas that were later included in the project. Overall, the feedback from the organization stakeholders and the customer was positive.

References

1 JavaScript Tutorial, http://www.w3schools.com/js/, retrieved Nov 23rd 2014.

2 Microsoft Support Lifecycle, http://support2.microsoft.com/lifecycle/?LN=en-us&c2=12905, retrieved Nov 23rd 2014.

3 Netscape and Sun announce JavaScript, Dec 4th 1995, https://web.archive.org/web/20070916144913/http://wp.netscape.com/newsref/pr/newsrelease67.html, retrieved Nov 23rd 2014.

4 Microsoft Internet Explorer 3.0 Beta Now Available, May 29th 1996, http://news.microsoft.com/1996/05/29/microsoft-internet-explorer-3-0-beta-now-available/, retrieved Nov 23rd 2014.

5 Standard ECMA-262, 1st edition, June 1997, http://www.ecma-international.org/publications/files/ECMA-ST-ARCH/ECMA-262,%201st%20edition,%20June%201997.pdf, retrieved Nov 23rd 2014.

6 Standard ECMA-262, 5.1 Edition, June 2011, http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf, retrieved Nov 23rd 2014.

7 O'Reilly Media: Shyam Seshadri and Brad Green, Sep 5th 2014, AngularJS: Up and Running - First Edition, E-book.

8 Bleeding Edge Press: Pam Selle, Tim Ruffles, Christopher Hiller, and Jamie White, Nov 2014, Choosing a JavaScript Framework, E-book.

9 Techopedia: Application Framework, http://www.techopedia.com/definition/6005/application-framework, retrieved Jan 11th 2015.

10 W3Schools: JavaScript Libraries, http://www.w3schools.com/js/js_libraries.asp, retrieved Jan 17th 2015.

11 Blog: Uri Shaked, Aug 18th 2014, AngularJS vs. Backbone.js vs. Ember.js, https://www.airpair.com/js/javascript-framework-comparison, retrieved Jan 18th 2015.

12 Backbone.js, http://backbonejs.org/, retrieved Jan 18th 2015.

13 Allen Holub, 2004, Holub on Patterns: Learning design Patterns by Looking at Code – Softcover reprint of the hardcover first edition 2004.

14 Dojo toolkit, http://dojotoolkit.org/, retrieved Jan 29th 2015.

15 Martin Fowler, Jul 18th 2006, GUI Architectures, http://martinfowler.com/eaaDev/uiArchs.html, retrieved Jan 31st 2015.

16 Manning: Brian Ford and Lukas Ruebbelke, Jan 30th 2015, AngularJS in Action - Version 9 (Manning Early Access Program).

17 Knockout.js, http://knockoutjs.com/, retrieved Feb 1st 2015.

18 CGI NetWiser Fact Sheet, 2014.

19 Eclipse desktop & web IDEs, https://eclipse.org/ide, retrieved Mar 3rd 2015.

20 Angular Best Practice for App Structure (Public), https://docs.google.com/document/d/1XXMvReO8-Awi1EZXAXS4PzDzdNvV6pGcuaF4Q9821Es/pub, retrieved Mar 5th 2015.

21 TodoMVC Examples Source Code at GitHub, https://github.com/tastejs/todomvc, retrieved Jan 29th 2015.

22 Google Trends on ember.js, angularjs, backbone.js, knockout js and dojo toolkit form Jan 2005 to Jan 2015, http://www.google.com/trends/explore?hl=en-US#q=ember.js,+angularjs,+backbone.js,+knockout+js,+dojo+toolkit&date=1/2005+121m&cmpt=q, retrieved Feb 3rd 2015.

JavaScript Framework and Library Basic Comparison

JS framework / library	Active [1]	IE8	Open jobs [2]	Website
AccDC	✓	✓	0	http://whatsock.com/
Agility.js	x	?	N/A	http://agilityjs.com/
Ample SDK	x	✓	N/A	http://www.amplesdk.com/
AngularJS	✓	✓ [3]	>20	https://angularjs.org/
Atoms.js	x	✓	N/A	http://atoms.azurewebsites.net/docs/
AuraJS	x	✓	N/A	http://aurajs.com/
Backbone.js	✓	✓	10	http://backbonejs.org/
batman.js	x	x	N/A	http://batmanjs.org/
CanJS	x	✓	N/A	http://canjs.com/
CupQ	x	?	N/A	https://github.com/mobilewish/cupQ
DHTMLX	✓	✓ [4]	0	http://dhtmlx.com/
Dojo toolkit	✓	✓	1	http://dojotoolkit.org/
Echo3	x	✓	N/A	http://echo.nextapp.com/site/
Ember.js	✓	✓	2	http://emberjs.com/
Enyo	✓	✓	1	http://enyojs.com/
Ext JS	✓	✓	0	http://www.sencha.com/products/extjs/
feathers	x	?	N/A	http://feathersjs.com/
Flight	x	✓	N/A	http://flightjs.github.io/
Google Web Toolkit	✓	✓	1	http://www.gwtproject.org/
Iconic	x	x	N/A	http://ionicframework.com/
IIO Engine	x	?	N/A	http://iioengine.com/
JavaScriptMVC	x	✓	N/A	http://www.javascriptmvc.com/
jQuery	✓	✓ [5]	>20	http://jquery.com/
Knockout	✓	✓	5	http://knockoutjs.com/
Kraken	x	✓	N/A	http://cferdinandi.github.io/kraken/
midori	x	✓	N/A	http://www.midorijs.com/
MochiKit	x	✓	N/A	http://mochi.github.io/mochikit/
MooTools	✓	✓	0	http://mootools.net/
PhoneJS	?	x	N/A	http://propertycross.com/phonejs/
Prototype	✓	✓	0	http://prototypejs.org/
pyjs	✓	✓	0	http://pyjs.org/
qooxdoo	✓	✓	0	http://qooxdoo.org/
Reactive Coffee	x	x	N/A	https://github.com/yang/reactive-coffee
Rialto Toolkit	x	✓	N/A	http://rialto.improve-technologies.com/factory/
Rico	x	✓	N/A	http://openrico.sourceforge.net/
Riot.js	x	?	N/A	https://muut.com/riotjs/

JS framework / library	Active [1)]	IE8	Open jobs [2)]	Website
Sammy.js	✓	✓	0	http://sammyjs.org/
Skel	x	✓	N/A	http://getskel.com/
SmartClient / SmartGWT	✓	✓	0	http://smartclient.com/
soma.js	✓	✓	0	http://somajs.github.io/somajs
Spine	✓	x	N/A	http://spinejs.com/
SproutCore	✓	✓	0	http://sproutcore.com/
Stapes.js	x	?	N/A	http://hay.github.io/stapes/
Underscore.js	✓	?	0	http://underscorejs.org/
Wakanda	✓	x	N/A	http://www.wakanda.org/
YUI	✓	✓	0	http://yuilibrary.com/
ZK	✓	✓	0	http://www.zkoss.org/
Webix	x	✓	N/A	http://webix.com/

[1)] Active for last three or more years
[2)] Open job positions in Finland monster.fi, oikotie.fi, duunitori.fi, te-palvelut.fi, linkedin.com (N/A = not checked)

[3)] Support dropped in 1.3.x version

[4)] Version not specifically mentioned

[5)] Support dropped in 2.x version

Data collected Nov 17th 2014